IBM Worklight V6.1.0

Contents

Chapter 1. Overview of IBM Worklight

Start here to learn about the 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 applications (apps). 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.

For more information about IBM Worklight, see the following topics.

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 flexible structure gives you more options when you implement your mobile communication channel, or release a new version of your application. You can evaluate the best approach for each situation according to skills, time, and functionality, without being limited by a specific approach to mobile application development.

Web development

With 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 run 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 specifics 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
Distribution with built-in mechanisms	No	Yes	Yes	Yes

Table 1. Comparison of mobile development approaches (continued)

Aspect	Web development	Hybrid development	Hybrid mixed development	Native development
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 mobile applications (apps). 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 IBM Worklight for testing mobile applications. With the mobile testing capabilities of IBM Mobile Test Workbench for IBM 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 mode

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 that is called JSONStore for data exchange and storage. With this feature, you can 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. With IBM Worklight, you can define custom security handlers for any access to this flow of data. Because any access to data of a mobile application must 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 to be 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 to detect problems.

In addition to reports that summarize 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.

Monitoring

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.

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 onto 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. The Application Center provides a repository for storing the mobile application files and a web-based console for managing that repository. The Application Center 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 the following 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.

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 editor, 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.

Note: Because Apache Cordova is preinstalled with IBM Worklight, do not download your own version of Apache Cordova.

- Use both native and standard web languages within the same application to achieve development efficiency and provide 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 IBM Worklight application packages and is in fact a collection of web applications, optionally packaged as an EAR (Enterprise Application ARchive) file, that runs 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 user 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 reports about user adoption and usage (number and frequency of users that are engaging with the server through the applications).
- Configure data collection rules for application-specific events.
- Export raw reporting data to be analyzed by the Business Intelligence 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, the Windows Phone 8 platform, and the BlackBerry OS 6 and 7 platform, but does not target mobile web applications. Windows Phone 7, Windows RT, and BlackBerry OS 10 are 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 or IBM SmartCloud Orchestrator. With this pattern, administrators and corporations can respond quickly to changes in the business environment 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 or IBM SmartCloud Orchestrator also gives access to best practices and built-in expertise, such as built-in scaling policies.

IBM Worklight editions

IBM Worklight is available to you in several editions: the IBM Worklight Developer Edition, the IBM Worklight Consumer Edition, and the IBM Worklight Enterprise Edition.

IBM Worklight Developer Edition

IBM Worklight Developer Edition is a free, non-warrantied 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 IBM 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 are available from IBM Passport Advantage®. The IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition contain the following components:

- A separate Worklight Studio component, which is available as an Eclipse plug-in (Eclipse P2 installation)
- A separate Worklight Server component, which is available as an IBM Installation Manager package

- A separate, optional, IBM Mobile Test Workbench for IBM Worklight component, which is available as an Eclipse plug-in (Eclipse P2 installation)

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

System requirements for IBM Worklight include operating systems, Eclipse versions, SDKs, and other software.

To identify the system requirements for this release of IBM Worklight, see the IBM Worklight and IBM Mobile Foundation detailed system requirements on the IBM Support Portal. The IBM Worklight system requirements include:

- The operating systems that support IBM Worklight, including mobile device operating systems
- The required 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 Mobile OS feature mapping for IBM Worklight technote on the IBM Support Portal lists the IBM Worklight features that are available on each of the platforms that IBM Worklight supports.

Chapter 2. What's new

This section details the new features and changes in IBM Worklight V6.1.0 and subsequent fix packs.

What's new in IBM Worklight V6.1.0.2

IBM Worklight V6.1.0.2 fixes issues that were identified in previous versions.

Inclusion of interim fix for IBM Mobile Test Workbench for Worklight

Interim fix 001 for IBM Mobile Test Workbench for Worklight V8.5.1.3 is incorporated into IBM Worklight V6.1.0.2. The interim fix supports iOS 7 and 7.1.

Documentation improvements

The IBM Worklight V6.1.0.2 user documentation is updated to include the following improvements:

- Instructions on how to configure Worklight BlackBerry 10 project to work with WebWorks SDK 2.0. See “Worklight BlackBerry 10 project with WebWorks SDK 2.0” on page 460
- Additional details added to the descriptions of the onSuccess and onFailure methods. See WLResponseListener.
- Updated instructions on configuring device single sign-on. See “Configuring device single sign-on” on page 630.
- Clarified the description of the BusyIndicator API class. See WL.BusyIndicator.
- Clarification that multicultural support in the Application Center includes only the mobile client. The Application Center console is not translated. See “Multicultural support in the Application Center” on page 14.

Connection policy of HTTP adapters

In IBM Worklight V6.1.0, starting with Fix Pack 2, the behavior of the **`maxRedirects`** attribute of the `<ConnectionPolicy>` element of the HTTP adapter changed, as described in“The <connectionPolicy> element of the HTTP adapter” on page 534.

List of fixes for IBM Worklight V6.1.0.2

For a complete list of issues that are fixed in IBM Worklight V6.1.0.2, see Version 6.1.0 Fix Pack 2.

What's new in IBM Worklight V6.1.0.1

IBM Worklight V6.1.0.1 fixes many problems that were identified in previous versions.

Client-side log collection

Starting with IBM Worklight V6.1.0.1, you can now capture and receive uploaded client-side logs. For more information, see “Client-side log capture” on page 689.

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 was 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 `init` method of the WL.Client class 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 837.

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*.

Documentation improvements

The IBM Worklight V6.1.0.1 user documentation is updated to include the following improvements:

- Clarification on how to use the WL.App.BackgroundHandler `hideView` handler to hide the application splash screen.
- Instructions on how to install IBM Worklight support for cloud deployment from the command line. See “Installing IBM Worklight support for cloud deployment from the command line” on page 817.
- Instructions on how to integrate IBM WebSphere® DataPower® with a cluster of Worklight Servers. See “Integrating IBM WebSphere DataPower with a cluster of Worklight Servers” on page 201.
- Instructions on how to change message text strings. See “The WL.ClientMessages object” on page 699 and “Globalization mechanisms in IBM Worklight” on page 646.
- Descriptions of the JNDI properties that can be configured for the Application Center. See “List of JNDI properties for the Application Center” on page 172.

- Instructions on how to use IBM WebSphere DataPower as a push notification proxy for IBM Worklight. See “Using WebSphere DataPower as a push notification proxy” on page 1022.

List of fixes for IBM Worklight V6.1.0.1

For a complete list of issues that are fixed in IBM Worklight V6.1.0.1, see Version 6.1.0 Fix Pack 1.

Upgrading from Worklight Server V6.1.0 to V6.1.0.x in a production environment

For more information about the specific installation instructions to upgrade to Worklight Server V6.1.0.x (fix pack), see “Upgrading from Worklight Server V6.1.0 to V6.1.0.x in a production environment” on page 258.

Upgrading from Worklight Server V6.1.0 to an interim fix in a production environment

For more information about the specific installation instructions to upgrade to an interim fix, see “Upgrading from Worklight Server V6.1.0 to an interim fix in a production environment” on page 259.

New tutorials and samples

The list of tutorials and samples is augmented with the following new tutorials and their companion samples:

- The module *Windows Phone 8 - Using native pages in hybrid applications*, under category 6, *Adding native functionality to hybrid applications with Apache Cordova*
- The module *Creating an application with IBM Worklight Application Framework*, under category 9, *Advanced topics*
- The module *Developing dynamic, collaborative mobile applications with MQ Telemetry Transport*, under category 11, *Integrating with other products*

For more information about these new tutorials and samples, and how you can get started with IBM Worklight, see Chapter 3, “Tutorials and samples,” on page 27.

What's new in IBM Worklight V6.1.0

This section details the new features and changes in IBM Worklight V6.1.0 compared to the previous version of this product.

Easier Worklight Server upgrades and migration

Starting with IBM Worklight V6.1.0, numerous changes are implemented in Worklight Server. These improvements make it possible for you to deploy and run apps that are developed in any supported version of Worklight Studio on Worklight Server V6.1.0 or higher.

In the past, upgrading to a new version of Worklight Server required the involvement of IT personnel (to install and configure the new server) and Worklight developers (to migrate their existing apps to work with the new server version). This cooperation was required to ensure that the Worklight files (`.war.wlapp`, and `.wladapter`) deployed to the server matched the run time components of the server.

Since IBM Worklight V6.1.0, there is a separation of the Worklight Studio and the Worklight Server upgrade lifecycles. This separation means that:

- It is possible to upgrade an instance of Worklight Server to version 6.1.0 without upgrading your existing applications to Worklight Studio version 6.1.0. For further information, see “Separation of lifecycle between Worklight Server and Worklight Studio” on page 221.
- It is possible to deploy server artifacts (project WAR files, apps, and adapter files) that are developed with any supported version of Worklight Studio (V5.0.5, V5.0.6.x, V6.0.0.x, and V6.1.0) to an instance of Worklight Server V6.1.0. Direct Update also continues to work.

Increased ability to integrate IBM Worklight in common environments and production systems

IBM Worklight V6.1.0 makes it easier for you to reuse the skills, resources, and existing processes of your enterprise as you embrace IBM MobileFirst. It increases your ability to integrate IBM Worklight in common enterprise environments and in your production systems so that you can handle several applications to satisfy your business demands securely.

Multicultural support for user facing text

With IBM Worklight V6.1.0, the following languages are supported in the IBM Worklight client runtime user interface:

- English
- Chinese (Simplified)
- Chinese (Traditional)
- French
- German
- Italian
- Japanese
- Korean
- Portuguese (Brazil)
- Russian
- Spanish

With this support, all the messages and labels that are visible in user interface components that are part of the IBM Worklight client runtime components are available in these languages. This availability makes it easier for application developers to create globalized native and hybrid applications.

Developers are still responsible for providing translation for application-specific texts.

For more information about the language support and how to develop globalized apps, see “Developing globalized hybrid applications” on page 635.

Multicultural support in the Application Center

With IBM Worklight V6.1.0, the user interface of the Application Center mobile client has been translated into the following languages:

- Chinese (simplified)
- Chinese (traditional)

- French
- German
- Italian
- Japanese
- Korean
- Brazilian Portuguese
- Russian
- Spanish

The user interface of the Application Center console has not been translated.

Deployment to a protected console

With IBM Worklight V6.1.0, when you try to deploy an application or an adapter from Worklight Studio to a secure console, IBM Worklight checks whether the console is protected. If the console is protected, IBM Worklight presents a dialog to the user that asks to enter the user name and password.

The provided credentials are stored in Eclipse secure storage.

You can configure this storage by using **Window** > **Preferences** > **General** > **Security** > **Secure Storage**, for reuse for the next deployment.

Ant tasks are also available, for you to add credentials, and automate your deployment.

For more information, see “The Run on Worklight Development Server command” on page 499.

New Server Configuration Tool to ease deployment of IBM Worklight projects

With IBM Worklight V6.1.0, you can deploy a Worklight project by using the Server Configuration Tool.

The introduction of the new Worklight Server Configuration Tool makes it easier for IT staff to install and configure a new release of Worklight Server, and to deploy applications to it.

For more information, see “Deploying, updating, and undeploying a Worklight Server by using the Server Configuration Tool” on page 715.

Additional cloud deployment option by using IBM SmartCloud® Orchestrator

With IBM Worklight V6.1.0, you can deploy to the cloud by using IBM SmartCloud Orchestrator as well as IBM PureApplication System.

For more information, see “Deploying to the cloud by using IBM PureApplication System and IBM SmartCloud Orchestrator” on page 815.

Easy upgradability of mobile apps for new version of mobile OS without server upgrade

With IBM Worklight V6.1.0, you can easily enable your IBM Worklight applications to work on a newly released version of an already supported mobile OS by using the existing production server. Before this release, you also required a new version of Worklight Server to support an application that is generated from the corresponding new version of Worklight Studio.

FIPS 140-2 Support

With IBM Worklight V6.1.0, Federal Information Processing Standards (FIPS) 140-2 compliant libraries can be used to encrypt and decrypt data at rest (data stored locally using JSONStore) and data in motion (data exchanged between the Worklight Client and Worklight Server using HTTPS).

For more information, see “FIPS 140-2 support” on page 925.

Encryption of Worklight Server configuration properties with WebSphere encryption mechanism

The configuration of a Worklight server is based on confidential information that includes several passwords.

With IBM Worklight V6.1.0, when you deploy Worklight Server to WebSphere Application Server, you can use standard WebSphere encryption mechanisms to ensure that passwords that are stored in the `worklight.properties` file are not written in clear text.

For more information, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784.

Support of Windows Phone 8 in the Application Center

With IBM Worklight V6.1.0, the Application Center now supports the distribution of Windows Phone 8 company applications in addition to iOS, Android, and BlackBerry 6 and 7 applications.

The distribution of Windows Phone company applications imposes the following requirements:

- The organization has a Windows Phone enterprise account that permits the signing and internal distribution of applications.
- The Application Center mobile client and all Windows Phone 8 applications in the catalog must be signed with the enterprise signature.
- Each user must enroll their Windows Phone device with the enrollment token that corresponds to the enterprise account; the Application Center simplifies the delivery of the enrollment token to mobile devices.

In the Application Center, the administrator can perform the following operations for the distribution of Windows Phone applications:

- Upload Windows Phone applications to the catalog.
- Upload and manage enrollment tokens, so that users can enroll their devices before they install the mobile client for Windows Phone 8.

For more information, see:

- Application enrollment tokens in Windows Phone 8
- Installing the client on Windows Phone 8
- Installing an application on a Windows Phone device

Ability to access apps from enterprise or public app stores in Application Center

With IBM Worklight V6.1.0, the Application Center now supports links to applications stored in Apple iTunes or Google play. Now, both company applications and public applications are available from the Application Center. Therefore, the productivity tools that employees should be using are available through the same enterprise app store. You are no longer restricted to company applications that are available only in the Application Center.

Public applications are installed by following a link from the Application Center catalog and then following the installation procedure in the corresponding supported public app store.

When you have installed an application from a public app store, you can review it locally, but these reviews are not visible in the public app store.

For more information, see:

- Adding an application from a public app store
- Installing applications through public app stores

Improved LDAP support for configuring the Application Center

With IBM Worklight V6.1.0, the Application Center now benefits from the following improvements:

- Strong LDAP authentication for defining access control for the users and groups who can install mobile applications
- With Liberty Profile V8.5.5, the ability to connect to a federated registry that uses several LDAP registries for mapping users and groups to Application Center roles

For more information, see:

- Managing users with LDAP
- LDAP with Liberty Profile

Java EE role-based authorization option in WebSphere login module

Java Platform, Enterprise Edition (Java EE) role-based authorization option in the IBM WebSphere Login Module: The WebSphere Login Module (`com.worklight.core.auth.ext.WebSphereLoginModule`) now has an optional `role` parameter to specify the Java EE role that is required by the authenticated user. This option enforces the role-based authorization of Worklight resources that are protected by the IBM WebSphere Login Module. This feature takes advantage of the standard Java EE roles that are defined and managed by the IBM WebSphere Application Server.

For more information, see “WASLTPAModule login module” on page 618.

Improved license reports

With IBM Worklight V6.1.0, license reporting was improved, and you can now perform the following operations:

- Track client devices for IBM Worklight Enterprise Edition license
- Track apps for IBM Worklight Consumer Edition license

For more information, see “License tracking” on page 1010.

Improved documentation

With IBM Worklight V6.1.0, the documentation continues to be improved to help users develop, install, configure, and administer Worklight applications.

- The documentation now contains an administrator-focused introduction to IBM Worklight Security, LTPA, and LTPA integration with IBM Worklight.

  For more information, see “Worklight Security and LTPA overview” on page 801.
- The documentation now contains more information about push notification.

  For more information, see “Push notification” on page 585.

Faster development of better apps

IBM Worklight V6.1.0 delivers significant productivity improvement for developing hybrid applications and consuming services from either hybrid or native applications.

UI-based API discovery for SOAP and SAP

In IBM Worklight V6.1.0, you can quickly create apps that interact with back-end systems. With the new services discovery wizard, you can produce back-end integration code in IBM Worklight adapters much faster. Using this wizard, developers can explore WSDL or SAP Netweaver Gateway services, and generate IBM Worklight adapters that can interact with these services. The wizard takes you through the services discovery process and automates the creation of adapters, therefore considerably reducing the amount of coding required. To know how to use the services discovery wizard to automatically generate adapters, see “Generating adapters with the services discovery wizard” on page 547

You can then consume these services by using the generated adapter directly from your application code, or with IBM Worklight Application Framework (Beta code) that further facilitates the creation of services-driven mobile applications. For more information about IBM Worklight Application Framework, see “Visual rapid application development (Beta code).”

Visual rapid application development (Beta code)

IBM Worklight V6.1.0 provides a new set of tools, IBM Worklight Application Framework (Beta code), to help you develop services-driven mobile applications. With the IBM Worklight Application Framework editor, you can quickly build Worklight hybrid applications that interact with services exposed through WSDL or SAP Netweaver Gateway. To configure a services-driven app with IBM Worklight Application Framework, you perform the following operations in a single editor:

- Create representations of back-end services.
- Define data objects.

- Map data objects to services.
- Define views, and map them to data objects.
- Configure interactions between the services, data objects, and views

For more information about how to use IBM Worklight Application Framework, see “Developing hybrid applications with IBM Worklight Application Framework” on page 425.

Faster hybrid and web app development with instant preview

In IBM Worklight V6.1.0, your hybrid Worklight applications can be tested and debugged much faster as the web code can be instantaneously previewed inside the desktop browser and devices. With this new feature, the mobile front-end developer can save changes to the web resources (html, JavaScript, or css files), and immediately see the result of the changes by refreshing the previewed application in the browser and devices, without the need to rebuild the final app.

For more information, see “The Preview command” on page 502.

Developing and sharing enterprise UI patterns

Extending IBM Worklight V6.0.0, which provides out-of-box patterns for mobile UI page construction in Rich Page Editor, you can now configure the tool to use your own patterns with IBM Worklight V6.1.0. You can develop UI patterns with HTML, CSS, and JavaScript files, and package them inside a predefined archive format, so that they can be distributed among IT teams to reuse in their Worklight projects with Worklight Studio.

For more information, see “Adding a UI pattern to a Pattern Project” on page 395.

Sharing reusable application components

In IBM Worklight V6.1.0, you can define reusable libraries called application components that are based on Worklight project resources. You can then reuse application components by importing them into the applications you develop.

For more information, see “Application components” on page 479.

Sharing reusable Worklight project templates

With the streamlined "Run As" menus in IBM Worklight V6.1.0, you can export Worklight projects to define Worklight project templates. You can then reuse these templates to create new Worklight projects.

For more information, see “IBM Worklight project templates” on page 494.

Improved deployment UI from Worklight Studio

With IBM Worklight V6.1.0, it is now easier to run tasks to build and test Worklight applications with the local server, a remote server, in the browser, or on a device.

For more information, see “Building and deploying in Worklight Studio” on page 497.

Improved functional testing

IBM Worklight V6.1.0 includes an improved version of IBM Mobile Test Workbench for Worklight that offers the following improvements:

- Support for testing Dojo Mobile applications in addition to JQuery Mobile
- Ability to test mobile web applications
- Ability to record a test for a hybrid application on one family of device (Android or iOS) and replay on the other
- Support of iOS 7

For more information, see Testing with IBM Worklight.

Simplified JavaScript client-side runtime framework

IBM Worklight V6.1.0 includes a simplified JavaScript client-side runtime framework, which offers the following benefits:

- Improvement of the start time of mobile web and desktop browser applications by including only the subset of the Worklight javaScript framework code that is relevant for these environments
- Reduction of the size of hybrid applications by including only JavaScript framework code that is supported for the specific environment other than Android and iOS.

Ability to create apps with Location Services on more platforms

The geo-location toolkit makes it possible for business actions to be triggered when users reach a point of interest, or when users enter or exit a region (geo-fencing), and the geo-location toolkit can run server-side logic to enable meaningful reaction to important geo events. In IBM Worklight V6.1.0, the geo-location services are now available for developing applications of the following types:

- Native iOS apps
- Native Android apps
- Hybrid Windows Phone 8 apps

Improved Mobile Browser Simulator to help test location-based apps

One of the challenges mobile application developers are facing is how to test location-based applications. Although field testing is the best way to simulate real life scenarios, it can take time and be costly, especially if considerable traveling is required. With IBM Worklight V6.1.0, the improved Mobile Browser Simulator now includes testing capabilities of location aware mobile applications that use versatile location information like GPS and WiFi. The improved Mobile Browser Simulator brings the following benefits:

- The Geolocation widget supports enhanced precision and integrates with Worklight location services in addition to the W3C Geolocation API.
- The Network widget supports managing visible and connected access points, including signal strengths. These integrate with Worklight location services.
- The Scenario widget supports testing the use of both Geolocation and WiFi information including overlapping access points, areas with no GPS coverage, and simulating a user moving throughout the environment.

For more information, see “Platform support for location services” on page 661 and “Testing hybrid location service applications with MBS” on page 679.

Ability to subscribe or unsubscribe to notifications by using SMS

New SMS capabilities in IBM Worklight V6.1.0

If you want to extend your services to a larger population of users, especially in emerging markets, you must ensure that they cater to a large feature-phone-user population. The feature phone users do not have access to mobile apps or rich UI on their devices. Even in the case of smart phone users, some data networks are either expensive or unreliable, and you need an alternative channel of interaction with the enterprise. SMS is an effective channel in these instances for providing instant notifications to the users and providing them with ways to interact with the enterprise by sending SMS requests. IBM Worklight V6.1.0 adds new and improved capabilities for SMS messaging.

Two-way SMS messaging

Enterprises that choose to offer some of their services via the SMS channel can now use the IBM Worklight mobile platform to enable SMS-based two-way communication. IBM Worklight now offers support for both Mobile Originated (MO) as well as Mobile Terminated (MT) SMS message flows. An SMS message originating from the mobile device is routed via an SMS gateway before reaching the Worklight Server. The incoming message is then handled by an IBM Worklight custom adapter that in turn creates the response message to be sent back to the device. The Worklight Server can integrate with SMS gateway providers via the HTTP API and is capable of sending as well as receiving SMS messages.

For more information, see Using two-way SMS communication.

Improved app performance over unreliable or slow networks

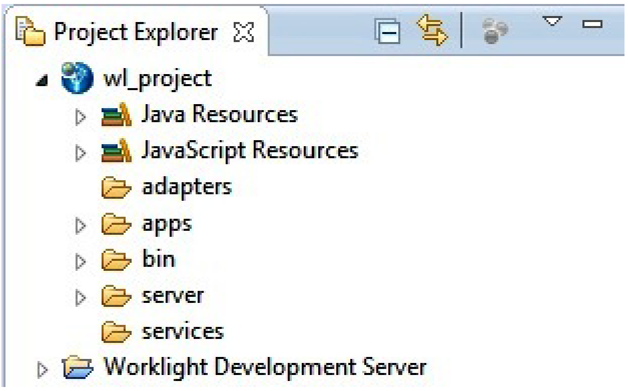
Slow and unreliable data networks can affect the app performance and the overall user experience. To deal with such network, IBM Worklight V6.1.0 offers the following improvements:

- Application Center: When the network connection is lost, and then reestablished, the application download can be resumed from where it stopped.
- Application Management: Direct Update of changes in web resources of a mobile app are safely delivered without disrupting the existing app. If the network connection is lost, and then reestablished, the user can resume the download from where it left off.
- Data Compression: JSON data that is returned from an adapter is compressed by default (based on a predefined threshold), which reduces the overall time that is taken to retrieve the back-end data. For more information, see “JSONStore performance” on page 579.

Simplified Worklight project structure in Worklight Studio

With IBM Worklight V6.1.0, the structure of a project is simplified, to focus on three main components that the user is interested in: adapters, apps, and services, as shown in the following figure.

Figure 1. Project Explorer showing simplified structure



Support for Android Studio

In IBM Worklight V6.1.0, developers are encouraged to use the latest Android SDK API level supported by Worklight Studio – Android 4.3 (API Level 18). Using the latest Android SDK allows the Android system to disable forward compatibility behaviors that slow the mobile application and to leverage the latest capabilities and features it includes. For more information, see the *Working with Android projects* section of "Migrating Worklight projects to Worklight Studio V6.1.0" on page 224.

With IBM Worklight V6.1.0, Android Studio V0.2.9 is now supported, in addition to support for Eclipse-based Android Developer Tools (ADT). Worklight Studio is now improved to integrate with Android Studio, the Android integrated development environment (IDE).

With the new Android integration, when you build, Worklight Studio can generate native Android project artifacts that are compatible with Android Studio. You can then open the Android Studio project inside a Worklight project in Worklight Studio in a similar way as Xcode projects or Visual Studio projects.

For more information, see "Additional Run As menu options" on page 506.

Improved mobile operations

IBM Worklight V6.1.0 brings improvements in terms of configuration, upgrade, installation, governance, and security.

Deprovisioning of lost device to stop service

IBM Worklight V6.1.0 comes with an optional feature to register each device to a user. When a device is registered, the administrator can use the Worklight Console to control that device's access rights to the Worklight Server. A device can be marked as stolen, lost, or disabled and will no longer be able to access resources on the Worklight server.

For more information, see "Mobile application management" on page 988.

Blocking access to apps

With IBM Worklight V6.1.0, the Worklight administrator can choose to block access for the entire device or for individual applications. When users should no longer access a specific app, the administrator can remove their access using the Worklight Console. For more information, see "Mobile application management" on page 988.

The default behavior when an app is remotely disabled in this manner is that the app can no longer be used by the device to access resources on the Worklight Server, although it can be run in offline mode. For information on how to completely disable an app so that the user is unable to even run it, see "Remotely disabling application connectivity" on page 837.

Clustering and operations of analytics engine

The IBM SmartCloud Analytics Embedded is built for handling large volumes of data. In this release, the IBM SmartCloud Analytics Embedded can be customized to operate in a clustered environment in order to scale for large amounts of data. For more information, see "IBM SmartCloud Analytics Embedded" on page 942.

User certificate authentication

X.509 certificates are typically provided by an MDM and are a costly solution. With the User Certificate Authentication feature, enterprises can now enroll users to their enterprise certificate authority (CA) and provision mobile devices with X.509 client-side certificates at a fraction of the cost.

With the User Certificate Authentication feature, enterprises can require users to authenticate and establish a secure HTTPS connection to the enterprise by using X.509 certificates that are issued to them by the enterprise PKI during an enrollment process. Users are no longer required to enter credentials every time they use an application. Additionally, access to the enterprise resources can now be entirely controlled by a PKI of choice.

For more information, see "User certificate authentication" on page 994.

Augmented support of mobile environments and operating systems

IBM Worklight V6.1.0 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 V6.1.0 has changed.

In particular, with IBM Worklight V6.1.0, the following environment is now supported:
- iOS 7

For more information about this list, see "System requirements for using IBM Worklight" on page 9.

Third-party libraries

IBM Worklight V6.1.0 is now based on Cordova 3.1. The upgrade process for Cordova configuration is automated.

Removed environments

With IBM Worklight V6.1.0, the following environments are now removed:
- Android V2.2. The minimum supported version is now Android V2.3.3.

- Windows Phone 7.5. The minimum supported version is now Windows Phone 8.

Enhanced IBM Worklight API

IBM Worklight V6.1.0 enhances and extends the API that you can use to develop mobile applications.

Support of updated libraries and toolkits

IBM Worklight V6.1.0 supports recent versions of the following libraries, which you can use during the development of your apps:

- Apache Cordova V3.1
- Dojo toolkit
- jQuery library
- jQuery Mobile library
- Sencha Touch framework

For more information, see “System requirements for using IBM Worklight” on page 9.

Updated JavaScript client-side API

IBM Worklight V6.1.0 includes some improvements in its JavaScript client-side API. Notice, in particular, the following ones:

- JSONStore API: Improved `count`, especially when there are more than 1000 documents in the collection, and you must decide how to set up your screen layout for optimal interaction. WL.JSONStore.count now supports a query.

  For more information, see the JSONStoreInstance class.

In IBM Worklight V6.1.0, the following JavaScript client-side API is deprecated:

- The JSONStore `fipsEnabled` option on `WL.JSONStore.init`: Deprecated.

  This option was described in the module *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for previous versions of IBM Worklight under category 5, *Advanced client side development*. The combination of the FIPS 140-2 and JSONStore optional features supersedes this option.

In IBM Worklight V6.1.0, the following elements of the JavaScript client-side API, which were deprecated in previous versions of IBM Worklight, are now removed:

- `WL.Page`: Removed
- `WL.Fragment`: Removed

Note: As a replacement to build your multi-page applications, consider using the equivalent implementation in JavaScript frameworks such as jQuery Mobile, Sencha Touch, and Dojo Mobile. You might also use embedded jQuery APIs to load page fragments (see http://api.jquery.com/load/). For more information about how to build a multi-page application, see the module *Building a multi-page application* under category 3, *Worklight client-side development basics*, in Chapter 3, “Tutorials and samples,” on page 27.

Updated JavaScript server-side API

IBM Worklight V6.1.0 includes some modifications in its JavaScript server-side API that you can use to extend the Worklight Server. Notice, in particular, that the API for geo-location services is now available for you.

For more information, see “JavaScript server-side API” on page 701.

Updated Objective-C client-side API for iOS

IBM Worklight V6.1.0 includes some modifications in its Objective-C client-side API to develop native apps on iOS. Notice, in particular, that the API for geo-location services is now available for you to develop native apps on iOS.

For more information, see “Objective-C client-side API for native iOS apps” on page 699.

Updated Java client-side API for Android

IBM Worklight V6.1.0 includes some modifications in its Java client-side API to develop native apps on Android. Notice, in particular, that the API for geo-location services is now available for you to develop native apps on Android.

For more information, see “Java client-side API for native Android apps” on page 699.

Miscellaneous modifications

IBM Worklight V6.1.0 includes other miscellaneous modifications, including changes in project files and behaviors and in the provided tutorials and samples.

Changes in file names

When you create an application with IBM Worklight V6.1.0, the main HTML, CSS, and JavaScript files of this new app are now called `index.html`, `main.css`, and `main.js` respectively.

Note: The file names of projects that were created with earlier versions of IBM Worklight do not change.

Enhanced tutorials and samples

In IBM Worklight V6.1.0, 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 27.

Chapter 3. Tutorials and samples

Tutorials and samples help you to get started with and learn about IBM Worklight, and evaluate what the product can do for you.

You can get started with IBM Worklight and learn how to develop mobile applications with IBM Worklight by studying the following tutorials and samples:

- “Tutorials”
- “Worklight Starter application samples” on page 35
- “JavaScript framework-based application samples” on page 35

Before you start developing mobile applications with IBM Worklight, consider reviewing the best practices at http://www.ibm.com/developerworks/mobile/worklight/best-practices.html to help guide your design and architecture decisions.

You can find links to download compressed files that contain the materials for the tutorials and samples in “Additional resources” on page 36.

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

Terms and conditions: The following resources are subject to these “Terms and conditions” on page 36, 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 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: The tutorials in this category describe how to set up your development environment to work with IBM Worklight.
2. Hello Worklight: The tutorials in this category describe how to create your first IBM Worklight app and preview it in different mobile operating systems.
3. Worklight client-side development basics: The tutorials in this category describe how to use basic IBM Worklight APIs to develop your apps, build a multi-page application, work with the user interface framework, and debug

and optimize your apps. Some general information is also provided that you must know to work in each specific environment.

4. Worklight server-side development: The tutorials in this category describe 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: The tutorials in this category describe how to implement different features in your mobile application, such as controls, skins, offline access, translation, and 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: The tutorials in this category describe how to use Apache Cordova with IBM Worklight, and how to use native pages in hybrid applications.
7. Developing native applications with Worklight: The tutorials in this category describe how to develop native applications with IBM Worklight.
8. Authentication and security: The tutorials in this category describe how to protect your applications and adapter procedures against unauthorized access by using authentication, login modules, and device provisioning.
9. Advanced topics: The tutorials in this category describe 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: The tutorials in this category describe how to move the apps that you create from your development environment to the production environment.
11. Integrating with other products: The tutorials in this category describe 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 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**
1. Setting up your development environment		
Setting up your Worklight development environment		This module explains how to set up your environment.
Setting up your iOS development environment		This module complements the module "Setting up your Worklight development environment " with further steps that are required for iOS application development.
Setting up your Android development environment		This module complements the module "Setting up your Worklight development environment " with further steps that are required for Android application development.
Setting up your BlackBerry 6 and 7 development environment		This module complements the module "Setting up your Worklight development environment " with further steps that are required for BlackBerry 6 and BlackBerry 7 application development.

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.
2. Hello Worklight		
Creating your first Worklight application	Exercise and code sample	This module explains how to set up your first mobile application.
Previewing your application on iOS		This module explains how to preview your application in the iOS environment.
Previewing your application on Android		This module explains how to preview your application in the Android environment.
Previewing your application on BlackBerry 6 and 7		This module explains how to preview your application in the BlackBerry 6 and BlackBerry 7 environments.
Previewing your application on BlackBerry 10		This module explains how to preview your application in the BlackBerry 10 environment.
Previewing your application on Windows Phone 8		This module explains how to preview your application in the Windows Phone 8 environment.
NEW: Previewing your application in Windows 8		This module explains how to preview your application in the Windows Phone 8 environment.
NEW: Previewing your application on Mobile Web and Desktop Browser		This module explains how to preview your application in the mobile web environment.
3. Worklight client-side development basics		
Learning Worklight client-side API	Exercise and code sample	This module explains the basics of the IBM Worklight Client API.
Building a multi-page application	Exercise and code sample	This module explains how to build a multi-page application with IBM Worklight.
Working with UI frameworks		This module explains how to work with the user interface (UI) frameworks of IBM Worklight.
Debugging your applications		This module explains how to debug the client applications.
Optimizing your application for various environments		This module explains how to optimize the application code for specific environments.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
General information when developing for iOS		This module gives some general information that you must know to develop apps for the iOS environment.
General information when developing for Android		This module gives some general information that you must know to develop apps for the Android environment.
General information when developing for BlackBerry 6 and 7		This module gives some general information that you must know to develop apps for the BlackBerry 6 and BlackBerry 7 environments.
General information when developing for BlackBerry 10		This module gives some general information that you must know to develop apps for the BlackBerry 10 environment.
General information when developing for Windows Phone 8		This module gives some general information that you must know to develop apps for the Windows Phone 8 environment.
General information when developing Mobile Web applications		This module gives some general information that you must know to develop mobile web applications.
General information when developing desktop applications		This module gives some general information that you must know to develop desktop applications.
4. Worklight server-side development		
Adapter framework overview		This module gives general information about adapters and the adapter framework in IBM Worklight, and how to work with adapters.
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. **Note:** This module has the same code sample as the module *HTTP adapter - Communicating with HTTP back-end systems*.
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). **Note:** This module has the same code sample as the module *HTTP adapter - Communicating with HTTP back-end systems*.

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. **Note:** This module has the same code sample as the module *HTTP adapter - Communicating with HTTP back-end systems*.
5. Advanced client-side development		
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 what actions are available to deal with the failures.
Enabling translation	Exercise and code sample	This module explains how to enable translation of the client applications.
Using Direct Update to quickly update your application		This module explains how to automatically update your applications with new versions of their web resources.
Storing sensitive data in encrypted cache	Exercise and code sample	This module explains how to work with the encrypted cache of the mobile device.
JSONStore - The client-side JSON-based database overview		This module introduces the JSONStore, and how you can work with JSON documents.
JSONStore - Common JSONStore Usage	Exercise and code sample	This module explains the common tasks that you can perform on a local JSON collection.
6. Adding native functionality to hybrid applications with Apache Cordova		
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.
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.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Android - Using native pages in hybrid applications	Exercise and code sample	This module explains how to use native pages in hybrid applications that are developed for the Android environment. **Note:** This module has the same code sample as the module *iOS - Using native pages in hybrid applications*.
Android - Adding native functionality to hybrid application with Apache Cordova plugin	Exercise and code sample	This module explains how to use Apache Cordova plug-ins to add native functionality to hybrid applications that are developed for the Android environment.
NEW: Windows Phone 8 - 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 Windows Phone 8 environment. **Note:** This module has the same code sample as the module *iOS - Using native pages in hybrid applications*.
Windows Phone 8 - Adding native functionality to hybrid application with Apache Cordova plugin	Exercise and code sample	This module explains how to use Apache Cordova plug-ins to add native functionality to hybrid applications that are developed for the Windows Phone 8 environment.
7. Developing native applications with Worklight		
Using IBM 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 how to use its components in a native iOS applications.
Using IBM 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 how to use its components in a native Android applications.
Using IBM 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 IBM 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 notifications in a native iOS applications.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
Using IBM 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 notifications in a native Android applications.
8. Authentication and security		
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.
Application Authenticity Protection		This module explains how to work with the application authenticity protection.
NEW: Client X.509 Certificate Authentication and User Enrollment	Exercise and code sample	This module explains how to work with the User Certificate Authentication feature by using the embedded PKI.
9. Advanced topics		
Shell development concepts	Exercise and code sample	This module explains the concepts that support the shell development and the inner applications.
Android shell development		This module explains how to develop Android applications by using shells.
iOS shell development		This module explains how to develop iOS applications by using shells.
Push notifications	Exercise and code sample	This module explains how to configure mobile devices to receive messages that are pushed from a server.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
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.
Two-way SMS communication	Exercise and code sample	This module explains how to communicate with Worklight Server from any mobile device by using the SMS channel.
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.
Location services	Exercise and code sample, Exercise and code sample (iOS), and Exercise and code sample (Android)	This module explains how to use geo-location services with IBM Worklight.
NEW: Creating an application with IBM Worklight Application Framework	Exercise and code sample	This module explains how to create an application with IBM Worklight Application Framework.
10. Moving to production		
Moving from development environment to stand-alone QA and production servers		This module explains how to move the components from the development environment into the test or production environment.
Operational Analytics		This module explains how you can use the operational analytics features of IBM Worklight as an alternative solution to BIRT reports.
Reports and analytics		This module explains the BIRT reports that help to collect the analytics data that pertains to applications and to devices that are accessing the Worklight Server.
Distributing mobile applications with Application Center		This module explains how you can distribute your apps with IBM Worklight Application Center.
11. Integrating with other products		
Using Rational Team Concert™ to build your applications	Exercise and code sample	This module explains how to develop as a team by using Rational® Team Concert.
Introducing Worklight Server and Application Center on IBM PureApplication System		This module explains 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 explains how you can integrate IBM Tivoli Directory Server with IBM PureApplication System.

Table 2. Getting Started modules and samples (continued)

Module	Sample (if any)	Description
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.
NEW: Accelerating application development by reusing resources		This module explains how you can use components and project templates with IBM Worklight.
Testing Worklight mobile applications with the Mobile Test Workbench		This module explains how to test your applications with IBM Worklight.
NEW: Developing dynamic, collaborative mobile applications with MQ Telemetry Transport	Exercise and code sample	This module explains how to develop dynamic, collaborative apps with MQ Telemetry Transport.

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. 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 release notes.

Globalization

If you are developing globalized apps, notice the following restrictions:

- Part of the product IBM Worklight V6.1.0, including its documentation, is translated in the following languages: Simplified Chinese, Traditional Chinese, French, German, Italian, Japanese, Korean, Portuguese (Brazil), Russian, and Spanish. Only user-facing text is translated.
- 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, adapters, Dojo custom builds and Dojo library projects must be composed only of the following characters:
  - Uppercase and lowercase letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)

  Some examples of the problems that you might encounter if the names of your Dojo library projects are NL strings are the incorrect display of the UI pattern preview, the failure of the generation of the Dojo custom build, and the failure to display the NL string in the Dojo custom build console.
- There is no support for Unicode characters outside the Basic Multilingual Plane.

The Server Configuration Tool has the following restrictions:

- The descriptive name of a server configuration can contain only characters that are in the system character set. On Windows, it is the ANSI character set.
- Passwords that contain single quotation mark or double quotation mark characters might not work correctly.

IBM SmartCloud Analytics Embedded 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 does 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 the `log` method as defined in the WL.Analytics class, or with the `error` method as defined in the WL.Logger class 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.
- 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.

Note: The standard Eclipse editor does not handle UTF-8 with the BOM (byte order mark) properly, therefore the Rich Page Editor does not support UTF-8 with byte order mark.

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 SmartCloud Analytics Embedded clustering. Consider testing your load prior to integrating IBM SmartCloud Analytics Embedded 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.

Worklight components cannot contain multiple environments

During the creation of a Worklight component, you include Android as well as iPhone or iOS environments. The addition of the same component fails because it contains multiple environments.

Separate Worklight component needs to be created for each environment, such as Android, iOS or iPhone.

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.

Installation of a fix pack or interim fix to the IBM Application Center or the Worklight Server

When you apply a fix pack or an interim fix to IBM Application Center or Worklight Server, manual operations are required, and you might have to shut down your applications for some time. For more information, see “Migrating from IBM Worklight V5.0.6 or later to V6.1.0” on page 221 or “Upgrading to Worklight Server V6.1.0 in a production environment” on page 226

Upgrade from Worklight Studio V5.0.0.3 to Worklight Studio V6.1.0

If you are using Worklight Studio V5.0.0.3 and you want to upgrade your projects to Worklight Studio V6.1.0, you must first migrate your projects to an intermediate version of the product, such as Worklight Studio V6.0.0. You can then migrate your projects from Worklight Studio V6.0.0 to Worklight Studio V6.1.0.

Note: This limitation applies only for Worklight Studio, and does not apply for Worklight Server.

FIPS 140-2 feature limitations

The following known limitations apply when using the FIPS 140-2 feature in IBM Worklight:

- This FIPS 140-2 validated mode applies only to the protection (encryption) of local data that is stored by the JSONStore feature and protection of HTTPS communications between the Worklight client and the Worklight Server.
  - For HTTPS communications, only the communications between the Worklight client and the Worklight Server use the FIPS 140-2 libraries on the client. Direct connections to other servers or services do not use the FIPS 140-2 libraries.
- This feature is only supported on the iOS and Android platforms.
  - On Android, this feature 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.
  - On iOS, it is supported on **`i386`**, **`armv7`**, and **`armv7s`** architectures.
- This feature works with hybrid applications only (not native).
- The use of the analytics feature (Tealeaf®) on the client is not supported by the FIPS 140-2 feature.
- The use of the user enrollment feature on the client is not supported by the FIPS 140-2 feature.
- The Worklight Application Center client does not support the FIPS 140-2 feature.
- The FIPS 140-2 feature supersedes the features described in the following module and article:
  - The module *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for previous versions of IBM Worklight under category 5, *Advanced client side development*.
  - The article *Enable FIPS 140-2 HTTPS encryption for IBM Worklight mobile apps* at http://www.ibm.com/developerworks/mobile/library/mo-fips-worklight/index.html.

  In both cases, you must delete the iPhone, iPad, and Android environments and re-create them.

  Important: When you delete the previous environments, you lose any environment-specific changes that you made. You must back up any environment-specific changes, and apply them to the newly created environment.
- The use of the Direct Update feature is not supported by the FIPS 140-2 feature.

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).

IBM Worklight Application Framework runtime library and Studio tools: beta code

You can add IBM Worklight Application Framework, as described in "Developing hybrid applications with IBM Worklight Application Framework" on page 425, to your IBM Worklight project or application. You use this feature for rapid development of hybrid Worklight applications. With IBM Worklight V6.1.0, the runtime library and the tooling support in Worklight Studio are beta code.

User Certificate Authentication feature limitations

The following known limitations apply when using the User Certificate Authentication feature in IBM Worklight:

- This feature is available only on the hybrid iOS and Android environments for this current release.
- This feature is not supported with the FIPS 140-2 feature.
- This feature is supported on WebSphere Application Server and WebSphere Application Server Liberty profile.
- This feature does not support an environment where the Worklight Server is protected by container security that requires a `CLIENT-CERT` authentication method. Instead, the server must be configured to accept the client certificate optionally, and not require one.
- Self-signed certificates are not supported with the User Certificate Authentication feature.

Chapter 5. Troubleshooting

You can find advice on how to troubleshoot problems, and more information about known limitations and Technote (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 51
- "Troubleshooting Worklight Server" on page 216
- "Troubleshooting an installation blocked by DB2 connection errors" on page 217
- "Troubleshooting failure to create the DB2 database" on page 217
- "Troubleshooting Worklight Development Server startup" on page 500
- "Troubleshooting IBM HTTP Server startup" on page 199
- "Troubleshooting to find the cause of installation failure" on page 216
- "Troubleshooting a Cast Iron adapter – connectivity issues" on page 529
- "Troubleshooting JSONStore and data synchronization" on page 584
- "Troubleshooting analytics" on page 966
- "Troubleshooting adding and removing application components" on page 494
- "Troubleshooting authenticity problems" on page 634
- "Troubleshooting the User Certificate Authentication feature" on page 1009
- "Troubleshooting a corrupt login page (Apache Tomcat)" on page 866
- "Troubleshooting failure to authenticate to Application Center and applications that use the basic registry element" on page 219

Important: For more information about known limitations or issues in the product, see Chapter 4, "Known limitations," on page 37 and the product release notes.

Important: If you have to contact IBM Support for help, see the information in Collect troubleshooting data for IBM Mobile Foundation and IBM Worklight products. This document details how to gather the necessary information about your environment so that IBM Support can help diagnose and resolve your problem.

Chapter 6. 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 components: Worklight Studio, Worklight Server, and IBM Mobile Test Workbench for Worklight. This section gives an overview of the installation process.

Installing Worklight Studio on Eclipse with a P2 update site

You install Worklight Studio into an existing installation of Eclipse by using its P2 install and update features. For actual instructions, see “Installing Worklight Studio” on page 46.

After the Worklight Studio installation, you must also install extra software development kits (SDKs) and Eclipse plug-ins for each mobile environment that you are developing for (for example, the Android Development Toolkit).

Installing IBM Mobile Test Workbench for Worklight on Eclipse with a P2 update site

You must install IBM Mobile Test Workbench for Worklight into an existing, properly configured installation of Worklight Studio by using the Eclipse P2 install and update feature. For actual instructions, see “Installing IBM Mobile Test Workbench for Worklight” on page 50.

Installing Worklight Server with IBM Installation Manager

To ensure the correct installation of Worklight Server, see “Installation prerequisites” on page 52.

You must install IBM Installation Manager 1.6.3 or later separately before installing IBM Worklight. For more information, see “Running IBM Installation Manager” on page 59.

Note: IBM Installation Manager is sometimes referred to on the eXtreme Leverage and Passport Advantage[®] sites and on the distribution disks as *IBM Rational Enterprise Deployment*. The filenames for the images take the form IBM `Rational Enterprise Deployment` *`<version number><hardware platform>`* *`<language>`*; for example, `IBM Rational Enterprise Deployment V1.6.3.1Windows Multilingual`.

You then use IBM Installation Manager to install Worklight server-side components on your application server, and to create databases on your database management system. Some application server and database configuration is required. For actual instructions, see “Installing Worklight Server” on page 52.

Upgrading IBM Worklight

The preceding sections provide an overview of IBM Worklight "first time" installations. For information about upgrading existing installations of IBM Worklight Studio and IBM Worklight Server to a newer version, see Chapter 7, “Upgrading from one version of IBM Worklight to another,” on page 221.

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.
- 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, as detailed in “System requirements for using IBM Worklight” on page 9. In particular, notice the minimum required version of Eclipse (errors happen if you use a previous version).
- 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 and verify your version of Eclipse. Worklight Studio V6.1.0 must be installed into Eclipse V4.2.2 (Juno) or Eclipse V4.3.1 (Kepler) as a new installation. It cannot be installed in earlier versions of Eclipse (prior to V4.2.2).
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. On the Available Software page, select **IBM Worklight Studio Development Tools**, and click **Next**. If you want to see the components to be installed, expand **IBM Worklight Studio Development Tools**, and select the components you want:
   - Always select **IBM Worklight Studio**.
   - Select **IBM Dojo Mobile Tools** if you anticipate using that JavaScript library.
   - Select **IBM jQuery Mobile Tools** if you anticipate using that JavaScript library.
8. On the Install Details page, review the features of Worklight Studio to be installed, and then click **Next**.
9. On the Review Licenses page, review the license text. If you agree to the terms, select **I accept the terms of the license agreement** and then click **Finish**.
10. The installation process starts. Follow the prompts (during which you may be asked to restart Eclipse) 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 27.

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. Alternatively, you can install Android Studio.

Procedure

- Using the Android SDK and the Android Development Tools plug-in for Eclipse
  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 27.

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.

- Using Android Studio
  1. Install Android Studio, available at http://developer.android.com/sdk/installing/studio.html.
  2. Update the Worklight Studio preferences with the location of Android Studio: from Worklight Studio, click **Window** > **Preferences** > **Worklight** (or **Eclipse** > **Preferences** > **Worklight** on Mac OS), and specify the location of the Android Studio installation.
  3. Right-click your Android applications, and click **Run As** > **Android Studio project** to start Android Studio.

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 27.

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 27.

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 9.
- 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.

Contents

Chapter 1. Installing the mobile test client

This section contains instructions for installing the IBM® Rational® Test Workbench Mobile Client.

Software and hardware requirements

Before you install the mobile test client and start working with the test workbench, be sure to understand the following requirements.

Operating system for the test workbench

- Microsoft Windows version 7, Vista, and 8
- Red Hat Linux version 5 and 6, SuSE Linux version 11
- Apple Mac OS X v.10.8 Mountain Lion or later (only supported in the IBM Worklight® environment)

Operating system for the device

- Android 2.2 - 4.3
- iOS 6.x and 7.x

JavaScript frameworks

- jQuery Mobile 1.3
- Dojo Mobile 1.9

The following jQuery widget is not supported in this release: Rangeslider

The following Dojo widgets are not supported in this release:

- Audio Player
- Video Player
- Time Picker
- Date Picker
- Gauges
- Map

Additional requirements for Android

- Android SDK 21 or 22

Additional requirements for iOS

- Mac OS X v10.8 Mountain Lion or later
- Xcode 4.6 or 5.0
- iOS Simulator 6.x or 7.x
- An Apple Developer or Enterprise License with a provisioning profile for each device. This is required for testing on iOS devices. It is not required for testing on the iOS Simulator.

Note: An Apple Macintosh computer is required to build and compile native and hybrid applications or to test with the iOS Simulator. A Macintosh computer is not required to test a web application on an iOS device or to test a native or hybrid application if you already have an instrumented version of the application under test.

Installing Android mobile test client

This section contains instructions for installing themobile test client for Android.

Installing the mobile test client on an Android device

To record and run tests from your Android mobile device, you must install the Android mobile test 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 be on the same network as the computer that is running the test workbench and be able to ping that computer. Furthermore, the mobile device must have a working internet connection (WiFi or cellular) or be tethered to the test workbench. 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. Otherwise, you can use the following method "Installing the mobile test client for Android with adb" on page 3.

Procedure

To install the mobile test client on the device:

1. In the Test Workbench perspective, click **File** > **New** > **Other** > **Test** > **Add Device**. A window displays a list of workbench URLs and a QR code that contains the selected 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. A notification is displayed when the download is completed. Open the notification and select the downloaded file, tap **Install** to install it.
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 the test workbench. See “Configuring the mobile test client for Android” on page 9.

Related concepts:

Mobile testing overview
The mobile testing capabilities of IBM Mobile Test Workbench for Worklight automate the creation, execution, and analysis of functional tests for native, web, and hybrid applications on Android and iOS devices.

Related tasks:

Getting started with Android testing
Use this topic to help you get started with your testing of applications that run on Android devices.

“Connecting an Android device to the test workbench with USB tethering” on page 5
The Android mobile client can be connected to the IBM Mobile Test Workbench for Worklight even if you have no WiFi connection. You can use a USB tethering and modify the settings on your device to enable this option.

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 2.

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 the test workbench. See “Configuring the mobile test client for Android” on page 9.

Related concepts:

Mobile testing overview
The mobile testing capabilities of IBM Mobile Test Workbench for Worklight automate the creation, execution, and analysis of functional tests for native, web, and hybrid applications on Android and iOS devices.

Related tasks:

Getting started with Android testing
Use this topic to help you get started with your testing of applications that run on Android devices.

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 2.

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 the test workbench. See "Configuring the mobile test client for Android" on page 9.

Related concepts:

Mobile testing overview
The mobile testing capabilities of IBM Mobile Test Workbench for Worklight automate the creation, execution, and analysis of functional tests for native, web, and hybrid applications on Android and iOS devices.

Related tasks:

Getting started with Android testing
Use this topic to help you get started with your testing of applications that run on Android devices.

Connecting an Android device to the test workbench with USB tethering

The Android mobile client can be connected to the IBM Mobile Test Workbench for Worklight even if you have no WiFi connection. You can use a USB tethering and modify the settings on your device to enable this option.

Before you begin

USB tethering is available on devices using Android 4.1 or later.

Procedure

1. Connect the Android device with the computer on which the Rational Test Workbenchtest workbench is running.
2. On the Android device go to **Settings** > **Wireless and networks**, and tap **More settings** and depending on the devices, tap **Mobile network sharing** > **USB network setting** or **tethering and hotspot settings**, then check **USB tethering** to share the mobile network with the PC.
3. On the computer make sure that all firewall windows are closed. Then click the icon to display the workbench URL and type the URL on the browser of the device to connect/reconnect to the test workbench. For more details on this step, see Configuring the mobile test client for Android

Uninstalling the Android mobile test client

Before uninstalling the mobile test client, you can reset it to uninstall the instrumented applications. When you uninstall the instrumented applications, you can use your original applications on your mobile device. Also, if you are upgrading to a new version of mobile test client, you will get updated version of instrumented applications.

Before you begin

It is recommended to reset the mobile test client to uninstall the instrumented applications. To reset, start mobile test client and from the menu, tap **Reset** and then **OK**.

Procedure

1. In the Android mobile device, tap **Settings** > **Apps** > **IBM Rational Test Workbench Mobile Client**.
2. Tap the **Uninstall** button and tap **OK**.

Installing iOS mobile test client

This section contains instructions for installing the IBM Rational Test Workbench Mobile Client for iOS.

Installing the mobile test client on the iOS Simulator

To record and run tests from your iOS Simulator, you must install the mobile test 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 the Test Workbench perspective, click **File** > **New** > **Other** > **Test** > **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 type the following command:

   ```
   ./installIPAInSimu.sh /Users/XXX/Downloads/RTW-iOS-Build-Archive/client/iphonesimulator/RTWiOS.ip
   ```

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

Related tasks:

Getting started with testing on the iOS Simulator
Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

Chapter 2. 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 mobile test client to connect to the test workbench.

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 2.

The mobile device must be on the same network as the computer that is running the test workbench and be able to ping that computer. Furthermore, the mobile device must have a working internet connection (WiFi or cellular) or be tethered to the test workbench. You can use an app that tests the connection to a computer.

Note: If you already have the workbench URL and the QR code up on your screen from when you installed the mobile test client, you can skip the first two steps and proceed to Step 3.

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 a list of workbench URLs and a QR code that contains the selected URL from the list.

   Note: If the device fails to connect using the first URL in the list, try one of the alternate URLs.
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 is the same QR code from when you installed the mobile test client. This step is not applicable to the emulator.
   b. Tap **Configure 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 the QR code button, or type the workbench address. These actions are executed automatically.

Results

The name and properties of the device are displayed in the test workbench, in the **Mobile Devices** editor.

Related tasks:

"Connecting an Android device to the test workbench with USB tethering" on page 5
The Android mobile client can be connected to the IBM Mobile Test Workbench for Worklight even if you have no WiFi connection. You can use a USB tethering and modify the settings on your device to enable this option.

Configuring the iOS mobile test client on the iOS Simulator

To upload recordings and run tests from the mobile test client, the mobile test client must be connected to the test workbench.

Before you begin

Note: This task applies to testing native applications from an iOS simulator.

Both the mobile test client and the test workbench must be installed and able to communicate with each other.

About this task

If the mobile test client is connected to the test workbench, the recording is automatically uploaded to the test workbench after recording the app on the device. 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, click the icon in the test workbench.

Related tasks:

"Installing the mobile test client on the iOS Simulator" on page 6
To record and run tests from your iOS Simulator, you must install the mobile test client on the Simulator.

Instrumenting iOS applications on the iOS Simulator
Once the mobile test client is installed on the iOS Simulator, you must run a supplied build script on an OS Xsystem to be able to instrument the iOS application under test. The instrumented app can be optionally pushed to the test workbench and installed on the iOS Simulator.

Recording tests from the iOS mobile test client
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

Index

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 **JDK** 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 IBM SmartCloud Analytics Embedded.

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 7, "Upgrading from one version of IBM Worklight to another," on page 221.

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 "Worklight Server installation process walkthrough" on page 53 are based on a simple installation scenario. For a complete description of the process of deploying a Worklight project, see Chapter 10, "Deploying IBM Worklight projects," on page 711.

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 in "System requirements for using IBM Worklight" on page 9.

Important: If a version of Worklight Server is already installed, review Chapter 7, "Upgrading from one version of IBM Worklight to another," on page 221 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 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.

Verify that the application server you selected provides support for your database.

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 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

Worklight Server installation process walkthrough

Learn about the Worklight Server installation process by walking through 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, if you have not already done so. You use Worklight Studio to create a simple IBM Worklight project, which you can then run on Worklight Server.

About this task

This task shows you how to install Worklight Server, based on a walkthrough of a simple configuration. It is designed as an overview, to show 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 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 subsequent description of your configuration to IBM Support more difficult.

This task involves installing the following components:

- An IBM WebSphere Application Server Liberty Core application server.
- A database management System (IBM DB2, Oracle, or MySQL).
- 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.
   a. Load the repository for IBM WebSphere Application Server Liberty Core in IBM Installation Manager and install the product.

      Note: IBM Installation Manager is sometimes referred to on the eXtreme Leverage and Passport Advantage sites and on the distribution disks as *IBM Rational Enterprise Deployment*. The filenames for the images take the form `IBM Rational Enterprise Deployment` *`<version number><hardware platform> <language>`*; for example, `IBM Rational Enterprise Deployment V1.6.3.1Windows Multilingual`.

      If you are not familiar with IBM Installation Manager, see step 4a of this procedure for more information on loading repositories. See also the IBM Installation Manager user documentation at http://pic.dhe.ibm.com/infocenter/install/v1r6/index.jsp.
   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 server 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 the database management system. You use this DBMS to install the Application Center and to deploy an IBM Worklight project and its console.
   a. If you use IBM DB2:
      1) The installer for IBM DB2 is provided as part of the package for IBM Worklight. Run the installer and follow the instructions.
      2) When, on Windows, you are asked whether to install the IBM Secure Shell Server for Windows, say `Yes`.
      3) In the next steps, you must have a secure shell server (such as, on Windows, the IBM Secure Shell Server for Windows or the Cygwin `openssh` package, or, on UNIX, the `sshd` daemon) installed and running so that the Worklight tools can create the required databases.
      4) Take note of the username and password for the DB2 administrator role. They will be needed in the next steps.
   b. If you use MySQL:
      1) Install MySQL on your computer.
      2) Take note of the username and password for the administrator.
         - By default for some installations, the administrator is `root` and there is no password.
         - If there is no password for the MySQL administrator in your installation, set a password for the administrator, following the instructions from the MySQL documentation.
   c. If you use Oracle:
      1) Install the Oracle database on your computer.
      2) Install an `ssh` shell on your computer (on Windows, install `cygwin` and the `openssh` package, as described at http://docs.oracle.com/cd/E25178_01/install.1111/e22624/preinstall_req_cygwin_ssh.htm).
      3) Launch the `ssh` server (on Windows, with administrator rights).
      4) In the next steps, you will need to have that secure shell server running.
4. Install Worklight Server.
   a. Add the Worklight Server repository in IBM Installation Manager:
      1) Download *IBM Worklight Enterprise Edition V6.1 zip of Installation Manager Repository for IBM Worklight Server Multiplatforms English.*
      2) Unzip the file on your disk.
      3) Launch IBM Installation Manager.
      4) Open the **File** > **Preferences** menu.
      5) In the Preferences dialog, click **Add Repositories**.
      6) Select the file `disk1/diskTag.inf` from the repository directory you unzipped.
      7) Click **OK** and close the Preferences dialog.
   b. 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 the name of the database management system you installed. Note that Apache Derby is not supported by the Server Configuration Tool used later in this walkthrough.

3) In the following database panels of the installer:
   - If you use IBM DB2:
     - In the **Database Server Properties** panel:
       - Enter `localhost` as the **host name**.
       - Select the `db2jcc4.jar` JAR file in the JDBC driver directory (located in *<DB2InstallDir>*`/Java`).
     - In the **Database Server Additional Properties** panel:
       - Select **Simple Mode**.
       - Enter a database user and password. This user must already exist.
     - In the Create Database panel:
       - Enter the name and password of a user account on the database server that has DB2 privilege `SYSADM` or `SYSCTRL`.
       - The installer creates the database.
   - If you use MySQL:
     - In the **Database Server Properties** panel:
       - Enter `localhost` as the **host name**.
       - Enter the name of the JDBC JAR file for MySQL.
     - In the **Database Server Additional Properties** panel:
       - Select **Simple Mode**.
       - Enter a database user and password. This user is already created by the installer.
     - In the Create Database panel:
       - Enter the name and password of a superuser account in your MySQL database server. The default superuser account is root.
       - The installer creates the database.
   - If you use Oracle:
     - In the **Database Server Properties** panel:
       - Enter `localhost` as the **host name**.
       - Enter the name of the JDBC JAR file for Oracle.
     - In the **Database Server Additional Properties** panel:
       - Select **Simple Mode**.
       - Enter a password for the user `APPCENTER`. This user is created by the installer.
       - The installer creates a database if it does not already exist.
     - In the **Create Database** panel:
       - For **Administrator Login Name and Passwords**, enter an administrator login name and password that can be used to run an `ssh` session. The default Oracle Administrator Login name is `oracle`.
       - If the database already exists, provide the password of the `SYSTEM` user that is used to create the user `APPCENTER`. If the database does not already exist, enter the passwords for the `SYS` and `SYSTEM` users that are created to manage the database.

        4) In the **Application Server Selection** panel, select **WebSphere Application Server**.
        5) In the **Application Server Configuration** panel, select the installation directory for IBM WebSphere Application Server Liberty Core that is installed in step 2.
        6) Select **simpleServer** as the server name.
        7) Install the product.

   The files that are described in “Distribution structure of Worklight Server” on page 78 are installed on your computer.

5. 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 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 138.
      - See “Distribution structure of Worklight Server” on page 78 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 82. 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.

6. 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 46.
        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** > **Run on Worklight Development Server**.

        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`.

7. Deploy the Worklight Console with the Server Configuration Tool.
    a. Start the IBM Worklight Server Configuration Tool.
        - On Linux:
            – Click the desktop menu **IBM Worklight Server** > **Server Configuration Tool**.
        - On Windows:
            – Click the Start menu **IBM Worklight Server** > **Server Configuration Tool**.
        - On Mac OS X:
            – In the Finder, double-click the file *`WL_INSTALL_DIR`*`/shortcuts/configuration-tool.sh`.

        Note: Worklight Server is not supported for production use on Mac OS X.
    b. Select **Create a New Server Environment**.
    c. Name the configuration `Hello Worklight Server`.
    d. In the **General** panel:
        1) Load the WAR File that you created in the previous step.
        2) Check **Create Worklight Server Shortcut** and enter a directory for the shortcuts. The resulting files will contain the URL to the Worklight Console.
    e. In the **Database Properties** panel:
        1) Select your database.
        2) Proceed as described in the *Install Worklight Server* section when you entered data to create the database for Application Center.
    f. In the **Application Server** panel:
        - Proceed as described in the *Install Worklight Server* section when you entered data to create the database for Application Center.
    g. When all the data is entered, click **Deploy**.
        - The log of the deployment operations appears in the console.
        - The **Configuration** appears in the tree view.
        - After the database operation is completed, a log file named `databases` appears in the tree view, under the **Configuration**.
        - After the deployment to the application server is complete, a log file named `install` appears in the tree view, under the **Configuration**.
8. Restart the Liberty server and open the Worklight Console.
    a. Go to the Liberty installation directory. Type the following command:
        - On Linux and UNIX systems:

```
bin/server stop simpleServer
```

        - On Windows systems:

```
bin\server.bat stop simpleServer
```

    b. Restart the server with the following command:

- On Linux and UNIX systems:

  ```
  bin/server start simpleServer
  ```
- On Windows systems:

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

c. In the shortcut directory that you specified in the **General** panel of the Server Configuration Tool:
   - On Linux and UNIX systems:

     Run the `worklight-console.sh` script.
   - On Windows systems:

     Double-click the file `worklight-console.url`. (On Windows 7, this shortcut can appear as `worklight-console`, with a file type of Internet Shortcut.)

   You should see the Worklight Console.

What to do next

For more information about the IBM Worklight Server Configuration Tool, see “Deploying, updating, and undeploying a Worklight Server by using the Server Configuration Tool” on page 715.

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 795.
- Review Chapter 11, “Administering IBM Worklight applications,” on page 833.
- Review “Deploying the project WAR file” on page 714.
- Review “Configuration of IBM Worklight applications on the server” on page 772 and “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784
- 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 767.

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.

Note: IBM Installation Manager is sometimes referred to on the eXtreme Leverage and Passport Advantage sites and on the distribution disks as *IBM Rational Enterprise Deployment*. The filenames for the images take the form IBM `Rational Enterprise Deployment <`*`version number`*`><`*`hardware platform`*`> <`*`language`*`>`; for example, `IBM Rational Enterprise Deployment V1.6.3.1Windows Multilingual`.

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 ~/sqllib/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 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 two different IBM Installation Manager modes.

Administrator installation

It is an administrator installation when IBM 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.

Nonadministrator (single-user) installation

It is a nonadministrator (single-user) installation when IBM Installation Manager is installed through the `userinst` command. In this case, only the user who installed this copy of IBM 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 nonadministrator (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 **Install**.
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 7, “Upgrading from one version of IBM Worklight to another,” on page 221. 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 **Install**.
     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 **Update**.

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 77.

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 complete a silent installation of Worklight Server on multiple computers or on computers 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 computers that are configured in the same way.
- You want to install Worklight Server on a computer where a GUI is not readily available. For example, a GUI might not be available on 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 that is called a response file. This file contains the necessary data to complete installation operations silently. Silent installations are started 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.

You can use a response file to do almost anything that is possible by using IBM 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 apply updates or interim fixes or to uninstall a package.

Silent installation is described in the IBM Installation Manager documentation, see Working in silent mode.

There are two ways to create a suitable response file:

- Working with sample response files provided in the IBM Worklight Information Center.
- Working with a response file recorded on a different computer.

Both of these methods are documented in the following sections.

In addition, for a list of the parameters that are created in the response file by the IBM Installation Manager wizard, see "Silent installation parameters" on page 70.

Working with sample response files for IBM Installation Manager:

Instructions for working with sample response files for IBM Installation Manager to facilitate creating a silent Worklight Server installation.

Procedure

Sample response files for IBM Installation Manager are provided in the `Silent_Install_Sample_Files.zip` compressed file. The following procedures describe how to make use of them.

1. Pick the appropriate sample response file from the compressed file. The `Silent_Install_Sample_Files.zip` file contains one subdirectory per release; for this release, you find the sample files in the subdirectory named `6.1`.

   For an installation that does not install IBM Application Center on an application server, use the file named `install-no-appcenter.xml`. For an installation that installs IBM Application Center, pick the sample response file from the following table, depending on your application server and database.

Table 6. Sample installation response files in the `Silent_Install_Sample_Files.zip` file

Application server	Derby	IBM DB2	MySQL	Oracle
WebSphere Application Server Liberty Profile	`install-liberty-derby.xml`	`install-liberty-db2.xml`	`install-liberty-mysql.xml` (see Note)	`install-liberty-oracle.xml`
WebSphere Application Server Full Profile, stand-alone server	`install-was-derby.xml`	`install-was-db2.xml`	`install-was-mysql.xml` (see Note)	`install-was-oracle.xml`

WebSphere Application Server Network Deployment	n/a	`install-wasnd-cluster-db2.xml`
		`install-wasnd-server-db2.xml`
		`install-wasnd-node-db2.xml`
		`install-wasnd-cell-db2.xml`
			`install-wasnd-cluster-mysql.xml` (see Note)
			`install-wasnd-server-mysql.xml` (see Note)
			`install-wasnd-node-mysql.xml`
			`install-wasnd-cell-mysql.xml` (see Note)
				install-wasnd-cluster-oracle.xml
				`install-wasnd-server-oracle.xml`
				`install-wasnd-node-oracle.xml`
				`install-wasnd-cell-oracle.xml`

Table 6. Sample installation response files in the `Silent_Install_Sample_Files.zip` file (continued)

Application server	Derby	IBM DB2	MySQL	Oracle
Apache Tomcat	`install-tomcat-derby.xml`	`install-tomcat-db2.xml`	`install-tomcat-mysql.xml`	`install-tomcat-oracle.xml`

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.

For uninstallation, use a sample file that depends on the version of Worklight Server that you initially installed in the particular package group. (Worklight Server V6.x and newer uses a package group named **IBM Worklight**, whereas Worklight Server V5.x used the package group **Worklight**.)

Table 7. Sample uninstallation response files in the `Silent_Install_Sample_Files.zip`

Initial version of Worklight Server	Sample file
V5.x	`uninstall-initially-v5.xml`
V6.x	`uninstall-initially-v6.xml`

2. Change the file access rights of the sample file to be as restrictive as possible. Step 4 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 <target-file.xml>
     ```
   - On Windows:
     ```
     cacls <target-file.xml> /P Administrators:F %USERDOMAIN%\%USERNAME%:F
     ```
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 WebSphere Application Server Liberty profile: `wlp/usr/servers/<server>/server.xml`
   - For Apache Tomcat: `conf/server.xml`
4. Adjust the list of repositories, in the `<server>` element. For more information about this step, see section named *Information about the repositories* in “Become familiar with IBM Installation Manager before you start” on page 236 and the IBM Installation Manager documentation at Repositories.

   In the `<profile>` element, adjust the values of each key/value pair.

   In the `<offering>` element in the `<install>` element, set the version attribute to match the release you want to install, or remove the version attribute if you want to install the newest version available in the repositories.
5. Perform the installation by using the `imcl` command, as described in the IBM Installation Manager documentation at Installing a package silently by using a response file.

Working with a response file recorded on a different machine:

Instructions for working with response files for IBM Installation Manager created on another machine to facilitate creating a silent Worklight Server installation.

Procedure

1. Record a response file, by running IBM Installation Manager in wizard mode and with option `-record` *`responseFile`* on a machine where a GUI is available. For more details, see Record a response file with Installation Manager. The following code example shows a recorded response file:

```
<?xml version="1.0" encoding="UTF-8"?>
<agent-input acceptLicense='true'>
  <server>
    <repository location='http://packages.example.com/ibm/worklight-5.0.5/'/>
  </server>
  <profile id='Worklight' installLocation='/opt/IBM/Worklight'>
    <data key='eclipseLocation' value='/opt/IBM/Worklight'/>
    <data key='user.import.profile' value='false'/>
    <data key='cic.selector.os' value='linux'/>
    <data key='cic.selector.ws' value='gtk'/>
    <data key='cic.selector.arch' value='x86'/>
    <data key='cic.selector.nl' value='en'/>
    <data key='user.writable.data.group' value='admin'/>
    <data key='user.database.db2.port' value='50000'/>
    <data key='user.database.preinstalled' value='true'/>
    <data key='user.database.selection' value='db2'/>
    <data key='user.database.db2.host' value='db2-101.example.com'/>
    <data key='user.database.db2.username' value='wl5test'/>
    <data key='user.database.db2.password' value='{xyzzy}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='main.feature'
            installFixes='none'/>
  </install>
  <preference name='com.ibm.cic.common.core.preferences.eclipseCache'
            value='/n/java/rational/SDP2Shared'/>
  <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='false'/>
  <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. Change the file access rights of the response file to be as restrictive as possible. Step 4 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 response-file.xml
  ```

- On Windows:

  ```
  cacls response-file.xml /P Administrators:F %USERDOMAIN%\%USERNAME%:F
  ```

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 WebSphere Application Server Liberty profile: wlp/usr/servers/<server>/server.xml
   - For Apache Tomcat: conf/server.xml
4. 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 5.

   Note: This is an example file based on the response file created 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 IBM 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 IBM 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%2Fguid
       <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='wl5test'/>
       <data key='user.database.db2.password' value='{xyzzy}7284OFD1KRHW8AC13S'/>
   ```

```
      <data key='user.database.db2.driver' value='/n/databases/drivers/db2-10.1/db2jcc4.jar'/>

      <!-- Application server choice. -->
      <data key='user.appserver.selection' value='was85liberty'/>
      <data key='user.appserver.was85liberty.preinstalled' value='false'/>

      <!-- Operating system group that shall be allowed to start the server. -->
      <data key='user.writable.data.group' value='admin'/>

    </profile>

    <!-- Define what IBM 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='main.feature'
                installFixes='none'/>
    </install>

    <!-- The IBM Installation Manager preferences don't need to be transferred to the
         target machine. -->

  </agent-input>
```

5. 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 8. Parameters available for silent installation

Key	When necessary	Description	Allowed values
`user.appserver.selection2`	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 **`user.appserver.selection2`** and **`user.database.selection2`** must take the value `none`.
`user.appserver.was.installdir`	${user.appserver.selection2} == was	WAS installation directory.	An absolute directory name.

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`user.appserver.was.profile`	${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.
`user.appserver.was.cell`	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	WAS cell into which to install the applications.	The name of the WAS cell.
`user.appserver.was.node`	${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.
`user.appserver.was.scope`	${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 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
user.appserver.was.serverInstance	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty && ${user.appserver.was.scope} == server	Name of WAS server into which to install the applications.	The name of a WAS server on the current machine.
user.appserver.was.nd.cluster	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty && ${user.appserver.was.scope} == nd-cluster	Name of WAS ND cluster into which to install the applications.	The name of a WAS ND cluster in the WAS cell.
user.appserver.was.nd.node	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty && (${user.appserver.was.scope} == nd-node || ${user.appserver.was.scope} == nd-server)	Name of WAS ND node into which to install the applications.	The name of a WAS ND node in the WAS cell.
user.appserver.was.nd.server	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty && ${user.appserver.was.scope} == nd-server	Name of WAS ND server into which to install the applications.	The name of a WAS ND server in the given WAS ND node.
user.appserver.was.admin.name	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Name of WAS administrator.	
user.appserver.was.admin.password2	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Password of WAS administrator, optionally encrypted in a specific way.	
user.appserver.was.appcenteradmin.password	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Password of `appcenteradmin` user to add to the WAS users list, optionally encrypted in a specific way.	
user.appserver.was.serial	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Suffix that distinguishes the applications to be installed from other installations of Worklight Server.	String of 10 decimal digits.

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
user.appserver.was85liberty.serverInstance	${user.appserver.selection2} == was && ${user.appserver.was.profile} == Liberty	Name of WAS Liberty Server into which to install the applications.	
user.appserver.tomcat.installdir	${user.appserver.selection2} == tomcat	Apache Tomcat installation directory. For a Tomcat installation that is split between a `CATALINA_HOME` directory and a `CATALINA_BASE` directory, here you need to specify the value of the **`CATALINA_BASE`** environment variable.	An absolute directory name.
user.database.selection2	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 **`user.appserver.selection2`** and **`user.database.selection2`** must take the value `none`.
user.database.preinstalled	Always	`true` means a preinstalled database management system, `false` means Apache Derby to install.	`true`, `false`

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`user.database.derby.datadir`	${user.database.selection2} == derby	The directory in which to create or assume the Derby databases.	An absolute directory name.
`user.database.db2.host`	${user.database.selection2} == db2	The host name or IP address of the DB2 database server.	
`user.database.db2.port`	${user.database.selection2} == db2	The port where the DB2 database server listens for JDBC connections. Usually 50000.	A number between 1 and 65535.
`user.database.db2.driver`	${user.database.selection2} == db2	The absolute file name of `db2jcc.jar` or `db2jcc4.jar`.	An absolute file name.
`user.database.db2.appcenter.username`	${user.database.selection2} == db2	The user name used to access the DB2 database for Application Center.	Non-empty.
`user.database.db2.appcenter.password`	${user.database.selection2} == db2	The password used to access the DB2 database for Application Center, optionally encrypted in a specific way.	Non-empty password.
`user.database.db2.appcenter.dbname`	${user.database.selection2} == db2	The name of the DB2 database for Application Center.	Non-empty; a valid DB2 database name.
`user.database.db2.appcenter.schema`	${user.database.selection2} == db2	The name of the schema for Application Center in the DB2 database.	

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`user.database.mysql.host`	${user.database.selection2} == mysql	The host name or IP address of the MySQL database server.	
`user.database.mysql.port`	${user.database.selection2} == mysql	The port where the MySQL database server listens for JDBC connections. Usually 3306.	A number between 1 and 65535.
`user.database.mysql.driver`	${user.database.selection2} == mysql	The absolute file name of `mysql-connector-java-5.*-bin.jar`.	An absolute file name.
`user.database.mysql.appcenter.username`	${user.database.selection2} == mysql	The user name used to access the MySQL database for Application Center.	Non-empty.
`user.database.mysql.appcenter.password`	${user.database.selection2} == mysql	The password used to access the MySQL database for Application Center, optionally encrypted in a specific way.	
`user.database.mysql.appcenter.dbname`	${user.database.selection2} == mysql	The name of the MySQL database for Application Center.	Non-empty, a valid MySQL database name.
`user.database.oracle.host`	${user.database.selection2} == oracle, unless ${user.database.oracle.appcenter.jdbc.url} is specified	The host name or IP address of the Oracle database server.	

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
user.database.oracle.port	${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.
user.database.oracle.driver	${user.database.selection2} == oracle	The absolute file name of `ojdbc6.jar`.	An absolute file name.
user.database.oracle.appcenter.username	${user.database.selection2} == oracle	The user name used to access the Oracle database for Application Center.	A string consisting of 1 to 30 characters: ASCII digits, ASCII uppercase and lowercase letters, '_', '#', '$' are allowed.
user.database.oracle.appcenter.username.jdbc	${user.database.selection2} == oracle	The user name used to access the Oracle database for Application Center, in a syntax suitable for JDBC.	Same as ${user.database.oracle.appcenter. if it starts with an alphabetic character and does not contain lowercase characters, otherwise it must be ${user.database.oracle.appcenter. surrounded by double quotes.
user.database.oracle.appcenter.password	${user.database.selection2} == oracle	The password used to access the Oracle database for Application Center, optionally encrypted in a specific way.	The password must be a string consisting of 1 to 30 characters: ASCII digits, ASCII uppercase and lowercase letters, '_', '#', '$' are allowed.

Table 8. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`user.database.oracle.appcenter.dbname`	${user.database.selection2} == oracle, unless ${user.database.oracle.appcenter.jdbc.url} is specified	The name of the Oracle database for Application Center.	Non-empty, a valid Oracle database name.
`user.database.oracle.appcenter.jdbc.url`	${user.database.selection2} == oracle, unless ${user.database.oracle.host}, ${user.database.oracle.port}, ${user.database.oracle.appcenter.dbname} are all specified	The JDBC URL of the Oracle database for Application Center.	A valid Oracle JDBC URL. Starts with "jdbc:oracle:".
`user.writable.data.user`	Always	The operating system user that is allowed to run the installed server.	An operating system user name, or empty.
`user.writable.data.group2`	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:

- When you are using WebSphere Application Server with DB2 as database type.
- When you are using WebSphere Application Server and have opened it without the application security enabled 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, restart the application server after the installation of Worklight Server completed.
- When 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 web applications are 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: Only the IBM Application Center is installed 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 10, "Deploying IBM Worklight projects," on page 711.

Distribution structure of Worklight Server

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

Table 9. Files and subdirectories in the Worklight Server installation directory

Item	**Description**
`shortcuts`	Launcher scripts for Apache Ant and the Worklight Server Configuration Tool, which are supplied with Worklight Server.

Table 10. Files and subdirectories in the `WorklightServer` subdirectory

Item	**Description**
`worklight-jee-library.jar`	The Worklight server library for production. See Chapter 10, "Deploying IBM Worklight projects," on page 711 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-builder.jar`	A set of Ant tasks that help you build projects, applications, and adapters for use in Worklight Server. See Chapter 10, "Deploying IBM Worklight projects," on page 711 for detailed documentation of the Ant tasks that are provided in this library.
`worklight-ant-deployer.jar`	A set of Ant tasks that help you deploy projects, applications, and adapters to your Worklight Server. See Chapter 10, "Deploying IBM Worklight projects," on page 711 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 762 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 732.

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

Item	Description
`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 779.
`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 982.

Table 11. Files and subdirectories in the `ApplicationCenter` subdirectory

Item	Description
`ApplicationCenter/installer`	**`IBMApplicationCenter.apk`** The Android version of the Application Center Mobile client. **`IBMApplicationCenterBB6.zip`** The Blackberry version of the Application Center Mobile client. **`IBMApplicationCenterUnsigned.xap`** The Windows Phone 8 version of the Application Center Mobile client. You must sign the `.xap` file with your company account before using it.
`ApplicationCenter/installer/IBMAppCenterBlackBerry6`	Contains the BlackBerry project for the mobile Client for OS V6 and V7. You must compile this project to create the BlackBerry version of the mobile client.
`ApplicationCenter/installer/IBMAppCenter`	Contains the Worklight Studio project for the mobile Client. You must compile this project to create the iOS version of the mobile client.
`ApplicationCenter/console/`	**`appcenterconsole.war`** The WAR file for the Application Center console user interface web application. **`applicationcenter.war`** The WAR file for the Application Center REST services web application. **`applicationcenter.ear`** The enterprise application archive (EAR) file to be deployed under IBM PureApplication System.

Table 11. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

Item	Description
`ApplicationCenter/databases`	**`create-appcenter-derby.sql`** The SQL script to recreate the application center database on derby. **`create-appcenter-db2.sql`** The SQL script to recreate the application center database on DB2. **`create-appcenter-mysql.sql`** The SQL script to recreate the application center database on mySQL. **`create-appcenter-oracle.sql`** The SQL script to recreate the application center database on Oracle. In addition, this directory contains the SQL scripts to upgrade the database from earlier versions of IBM Worklight.

Table 11. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

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

Table 12. Files and subdirectories in the `License` subdirectory

Item	Description
`License-wce`	License for IBM Worklight Consumer Edition
`License-wee`	License for IBM Worklight Enterprise Edition

Table 13. Files and subdirectories in the `tools` subdirectory

Item	Description
`tools/apache-ant-<version>`	A binary installation of Apache Ant that can be used to run the ant tasks described at Chapter 10, “Deploying IBM Worklight projects,” on page 711.

Table 14. Files and subdirectories in the `Analytics` subdirectory

Item	Description
`analytics_aix_ppc64.zip`	Archive for IBM SmartCloud Analytics Embedded for AIX®. See “Installing and configuring IBM SmartCloud Analytics Embedded” on page 213 for instructions to install and configure this platform.
`analytics_linux_x86_64.zip`	Archive for IBM SmartCloud Analytics Embedded for Redhat Linux. See “Installing and configuring IBM SmartCloud Analytics Embedded” on page 213 for instructions to install and configure this platform.

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 273) 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 60.
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 the following list shows.

Setting up your DB2 database manually for Application Center:

You can set up your DB2 database for Application Center manually.

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, go to ~/sqllib/bin and enter ./db2.
3. Enter the following database manager and SQL statements to create a database that is called **APPCNTR**:
   ```
   CREATE DATABASE APPCNTR COLLATE USING SYSTEM PAGESIZE 32768
   CONNECT TO APPCNTR
   GRANT CONNECT ON DATABASE TO USER worklight
   QUIT
   ```
4. Run 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. Determine a suitable directory for the JDBC driver JAR file in the WebSphere Application Server installation directory.
   - For a standalone server, you can use a directory such as `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/db2`.
   - For deployment to a WebSphere Application Server ND cell, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/Worklight/db2`.
   - For deployment to a WebSphere Application Server ND cluster, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/Worklight/db2`.
   - For deployment to a WebSphere Application Server ND node, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/Worklight/db2`.
   - For deployment to a WebSphere Application Server ND server, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/Worklight/db2`.

   If this directory does not exist, create it.
2. 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) and its associated license files, if any, to the directory determined in step 1.
3. 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 Using IBM JCC Driver**.
   e. Set **Implementation Type** to **Connection pool data source**.

f. Set **Name** to **DB2 Using IBM JCC Driver**.

g. Click **Next**.

h. Set the class path to the set of JAR files in the directory determined in step 1, replacing `WAS_INSTALL_DIR/profiles/`*`profile-name`* with the WebSphere Application Server variable reference **`${USER_INSTALL_ROOT}`**.

i. Do not set **Native library path**.

j. Click **Next**.

k. Click **Finish**.

l. The JDBC provider is created.

m. Click **Save**.

4. 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).

5. 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"
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 87
   - "Configuring WebSphere Application Server for Derby manually for Application Center" on page 87
   - "Configuring Apache Tomcat for Derby manually for Application Center" on page 89

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.EmbeddedConnectionPoolData
  <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. Determine a suitable directory for the JDBC driver JAR file in the WebSphere Application Server installation directory.
   - For a standalone server, you can use a directory such as `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/derby`.
   - For deployment to a WebSphere Application Server ND cell, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/Worklight/derby`.
   - For deployment to a WebSphere Application Server ND cluster, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/Worklight/derby`.
   - For deployment to a WebSphere Application Server ND node, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/Worklight/derby`.

- For deployment to a WebSphere Application Server ND server, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/Worklight/derby`.

If this directory does not exist, create it.

2. Add the `Derby JAR` file from `WORKLIGHT_INSTALL_DIR/ApplicationCenter/tools/lib/derby.jar` to the directory determined in step 1.
3. 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 the JAR file in the directory determined in step 1, replacing `WAS_INSTALL_DIR/profiles/`*`profile-name`* with the WebSphere Application Server variable reference **`${USER_INSTALL_ROOT}`**.
   j. Click **Finish**.
4. 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 86.
   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 61.
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 the following list shows.

Setting up your MySQL database manually for Application Center:

You can set up your MySQL database for Application Center manually.

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 the option -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 "at" 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.

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.

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.

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.

Procedure

1. Determine a suitable directory for the JDBC driver JAR file in the WebSphere Application Server installation directory.
   - For a standalone server, you can use a directory such as `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/mysql`.
   - For deployment to a WebSphere Application Server ND cell, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/Worklight/mysql`.
   - For deployment to a WebSphere Application Server ND cluster, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/Worklight/mysql`.
   - For deployment to a WebSphere Application Server ND node, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/Worklight/mysql`.
   - For deployment to a WebSphere Application Server ND server, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/Worklight/mysql`.

   If this directory does not exist, create it.
2. Add the MySQL JDBC driver JAR file downloaded from Download Connector/J to the directory determined in step 1.
3. Set up the JDBC provider:
   1. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   2. Create a **JDBC provider** named **MySQL**.
   3. Set **Database type** to **User defined**.
   4. Set **Scope** to **Cell**.
   5. Set **Implementation class** to **com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource**.
   6. Set **Database classpath** to the JAR file in the directory determined in step 1, replacing `WAS_INSTALL_DIR/profiles/`*`profile-name`* with the WebSphere Application Server variable reference **`${USER_INSTALL_ROOT}`**.
   7. Save your changes.
4. Create a data source for the IBM Application Center database:

   a. Click **New** to create a data source.
   b. Type any name (for example, `Application Center Database`).
   c. Set **JNDI Name** to `jdbc/AppCenterDS`.
   d. Use the existing **JDBC Provider MySQL**, defined in the previous step.
   e. Set Scope to **New**.
   f. On the **Configuration** tab, select **Non-transactional data source**.
   g. Click **Next** a number of times, leaving all other settings as defaults.
   h. Save your changes.

5. 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
```

6. 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 **Non-transactional data source**.
   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 61.
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 the following list shows.

Setting up your Oracle database manually for Application Center:

You can set up your Oracle database for Application Center manually.

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, specify **Unicode (AL32UTF8)** as the character set of the database.

   If the Oracle installation is on a UNIX or Linux machine, make sure that the database will be started the next time the Oracle installation is restarted. To this effect, make sure the line in `/etc/oratab` that corresponds to the database ends with a `Y`, not with an `N`.
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;
   ```

   Note: Access to a `TABLESPACE` is required for the user. You can replace `UNLIMITED TABLESPACE` by another `TABLESPACE` privilege.

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. Determine a suitable directory for the JDBC driver JAR file in the WebSphere Application Server installation directory.
   - For a standalone server, you can use a directory such as `WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight/oracle`.
   - For deployment to a WebSphere Application Server ND cell, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/Worklight/oracle`.
   - For deployment to a WebSphere Application Server ND cluster, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/Worklight/oracle`.
   - For deployment to a WebSphere Application Server ND node, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/Worklight/oracle`.
   - For deployment to a WebSphere Application Server ND server, use `WAS_INSTALL_DIR/profiles/`*`profile-name`*`/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/Worklight/oracle`.

   If this directory does not exist, create it.
2. Add the Oracle ⌂`ojdbc6.jar` file downloaded from JDBC and Universal Connection Pool (UCP) to the directory determined in step 1.
3. 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 15. 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 to the JAR file in the directory determined in step 1, replacing `WAS_INSTALL_DIR/profiles/`*`profile-name`* with the WebSphere Application Server variable reference **`${USER_INSTALL_ROOT}`**
   f. Click **Next**.

   The JDBC provider is created.
4. 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 82, 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-tools package. -->
   <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 140.

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 **Enable administrative security** is selected. Application security can only be enabled if administrative security is enabled.
   c. Ensure that **Enable application security** 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.
3. Create the Application Center JDBC data source and provider. See the instructions in the appropriate subsection in “Manually installing Application Center” on page 82.
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, click **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 modulespage.
   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 **Select** 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. Review the server class loader policy: Click **Servers** > **Server Types** > **Application Servers** > **Worklight**
   - If the class loader policy is set to **Multiple**, do nothing.
   - If the class loader policy is set to **Single** and the class loading mode is set to **parent-last**, do nothing.
   - If the class loader policy is set to **Single** and the class loading mode is set to **parent-first**, change the class loader policy to **Multiple**, and set the class loader order of all applications other than Worklight applications to **parent-first**.
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 139.

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 82.
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/realm-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 85
- "Configuring Apache Tomcat for Derby manually for Application Center" on page 89
- "Configuring Apache Tomcat for MySQL manually for Application Center" on page 92
- "Configuring Apache Tomcat for Oracle manually for Application Center" on page 96

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\a
     ```
4. Start Tomcat.

What to do next

For additional steps to configure the Application Center, see "Configuring Apache Tomcat" on page 141.

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.
- `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”).

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 x 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 that 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 772 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 779.

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

You can secure IBM Worklight in a typical WebSphere Application Server runtime environment in two ways.

The WebSphere Application Server provides Java Platform, Enterprise Edition container and server security by supporting various 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 that is running on WebSphere Application Server shown in the following diagram.

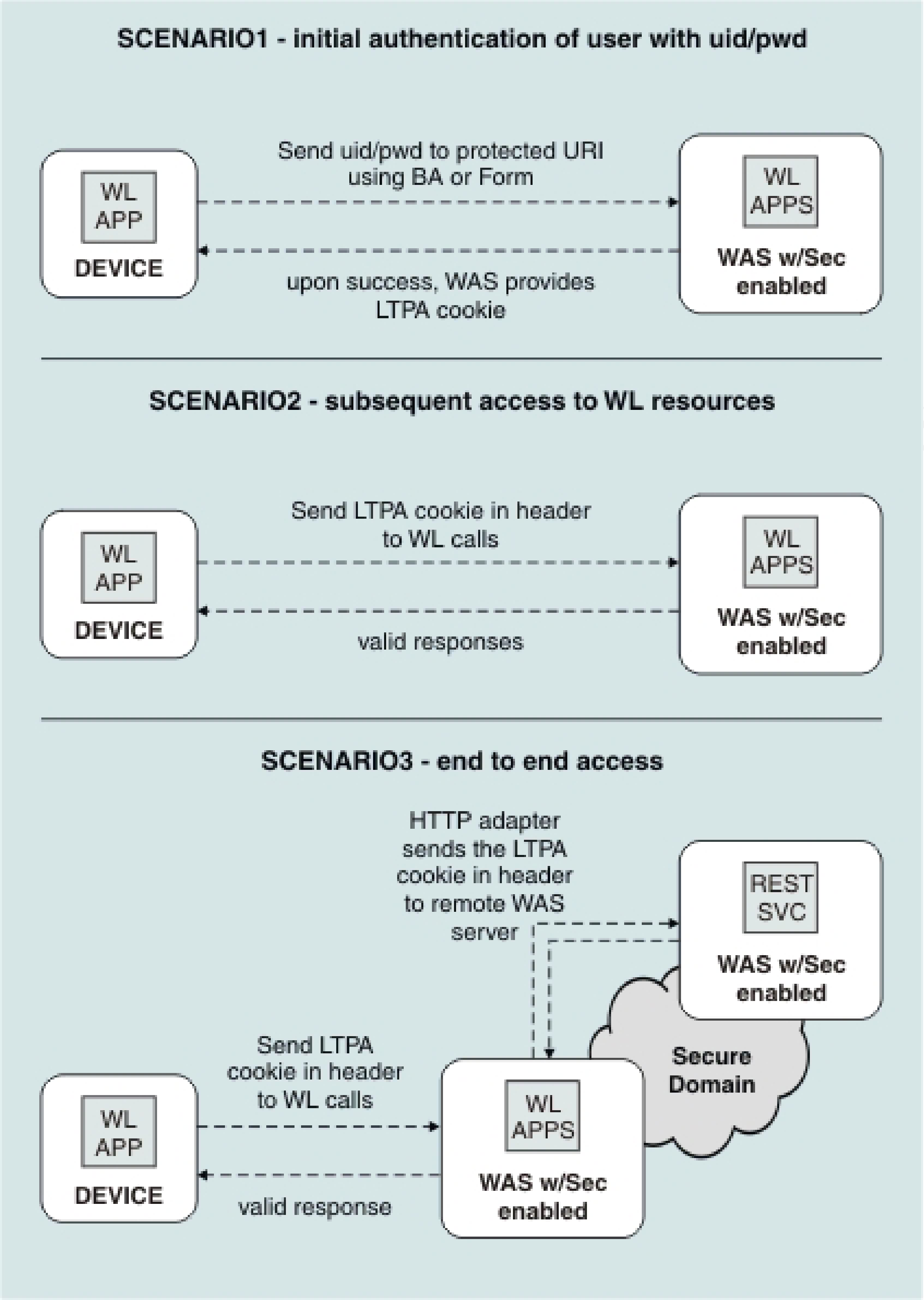

SCENARIO1 - initial authentication of user with uid/pwd
WL APP
DEVICE
Send uid/pwd to protected URI using BA or Form
upon success, WAS provides LTPA cookie
WL APPS
WAS w/Sec enabled
SCENARIO2 - subsequent access to WL resources
WL APP
DEVICE
Send LTPA cookie in header to WL calls
valid responses
WL APPS
WAS w/Sec enabled
SCENARIO3 - end to end access
HTTP adapter sends the LTPA cookie in header to remote WAS server
REST SVC
WAS w/Sec enabled
Secure Domain
WL APP
DEVICE
Send LTPA cookie in header to WL calls
valid response
WL APPS
WAS w/Sec enabled

Figure 2. Phases in securing an IBM Worklight app

When the user first accesses the app, the app prompts 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 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 by using application security and securing the IBM Worklight WAR file.
- Option 2: Securing WebSphere Application Server by 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 that is based on your specific requirements.

Option 1

Benefits

- Uses the traditional WebSphere Application Server authentication and trust model.
- The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container.

Usage

- Suitable for scenarios where the devices can be trusted and access for rogue applications is restricted.

Option 2

Benefits

- Uses the traditional WebSphere Application Server authentication and trust model without the impact of modifying the IBM Worklight Project WAR.
- The container enforces all security, so it can use existing third-party SSO products to secure the Java Platform, Enterprise Edition container.
- The layered authentication of device, application, application instance, and user functions as intended.
- Flexibility in configuring specific security settings that are specific to the IBM Worklight runtime environment without being hindered by the underlying container security.

Usage

- Suitable for scenarios where the devices or the apps on the devices cannot be trusted. The multi-step authenticity check that is built into IBM Worklight ensures denial of service to devices subjected to unauthorized modifications, rogue applications, and unauthorized users.

For more information about supported configurations for LTPA, see "Supported configurations for LTPA" on page 807.

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.</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 run 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 Worklight Server in WebSphere Application Server with Java 2 security enabled

You can run Worklight Server in WebSphere Application Server with Java 2 security enabled.

About this task

To enable Java 2 security in WebSphere Application Server, 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 Worklight Server on a WebSphere Application Server instance. The installation 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 **Use Java 2 security to restrict application access to local resources**.
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.

   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";
  // In Linux need to set TEMP folder of Linux.
  permission java.io.FilePermission "C:/Windows/TEMP/${/}-", "read,write,delete";
  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";
};
```

4. 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 3 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.

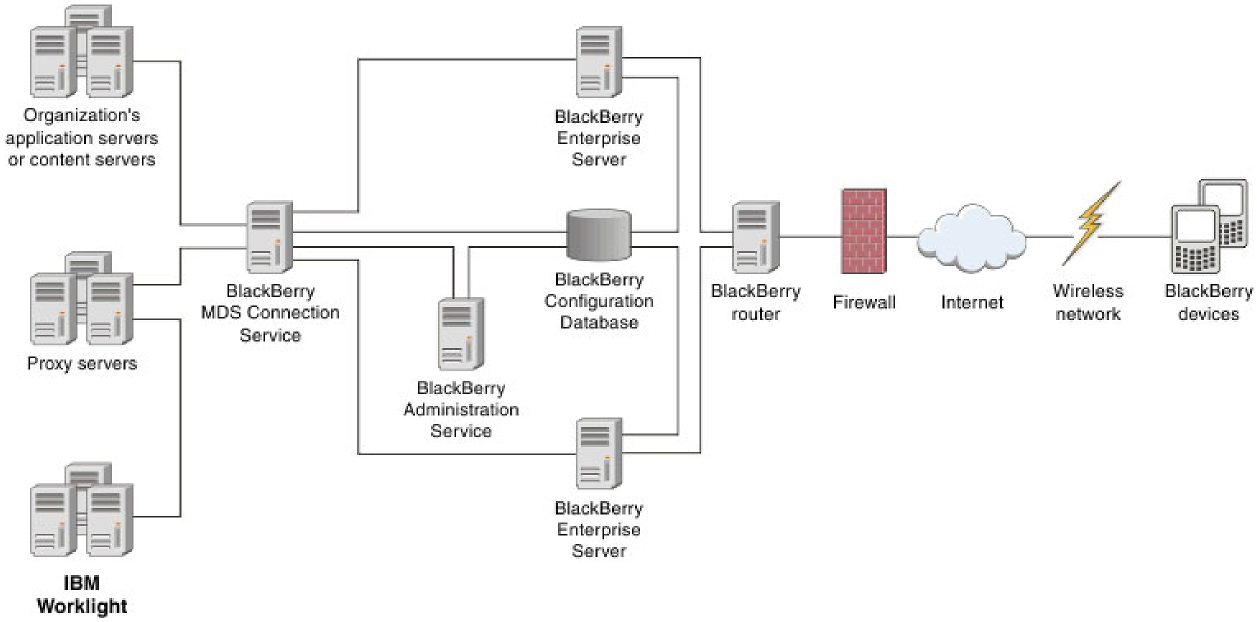

Organization's application servers or content servers
Proxy servers
IBM Worklight
BlackBerry MDS Connection Service
BlackBerry Enterprise Server
BlackBerry Configuration Database
BlackBerry Administration Service
BlackBerry Enterprise Server
BlackBerry router
Firewall
Internet
Wireless network
BlackBerry devices

Figure 3. 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 IBM 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="WASTest-securityTest">
    <testDeviceId provisioningType="none"/>
    <testUser realm="WASLTPARealm"/>
  </mobileSecurityTest>
  <webSecurityTest name="WASTest-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.submitLoginFormCal

});
```

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 `login` method as defined in the WL.Client class:
`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.isUserAuthenti
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, mp
  </security>
</ipad>
```

3. Define a multi-protocol gateway.

a. In the IBM DataPower WebGUI, in the search box under Control Panel, enter `Multi-Protocol`, and then click **New Multi-Protocol Gateway**.

b. On the **General Configuration** page, provide the following details:

Table 16. General Configuration

Field	**Description**
Multi-Protocol Gateway Name	Provide a name for your gateway.
Response Type	Select **Non-XML**. This allows HTTP Web application traffic (including JSON, JavaScript, and CSS) to pass through the appliance.
Request Type	Select **Non-XML**. This allows HTTP Web application requests to be handled by the appliance.
Front Side Protocol	Select **HTTPS (SSL)**. 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: **Name** Enter a name for the configuration. **Port Number** 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 18 on page 117.) **Allowed Methods and Versions** Select **GET method** to enable support for HTTP Get. **SSL Proxy** Select an SSL Reverse Proxy profile to identify the SSL server.
Multi-Protocol Gateway Policy	Click **+**, and then create rules to define the policies listed in the following topics depending on the type of authentication you decide to use: • Policy `worklight-basicauth` for HTTP basic authentication: see "Rules for HTTP basic authentication" on page 118. • Policy `mpgw-form` for HTML form-based login authentication: see "Rules for HTML forms-based authentication" on page 119.
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 **Add**.
   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 17. AAA policy for HTTP basic authentication

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

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

Table 18. AAA policy for HTML forms-based authentication

Phase	Description
Extract Identity	In the **Methods** field, select **HTML Forms-based Authentication**. Select or create an HTML forms-based policy that has the **Use SSL for Login** option enabled, assigns **SSL Port** to the port number on which the MPGW is listening (that was specified in step 3), and has the **Enable Session Migration** option disabled.
Authenticate	Choose the authentication method. If WebSphere Application Server is using LDAP, configure LDAP here.
Extract Resource	Select **URL Sent by Client**.
Post processing	Generate an LTPA token. Specify **LTPA Token Expiry**, **LTPA Key File**, and **LTPA Key File Password**.

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

Table 19. Advanced settings

Field	Value
Persistent Connections	**On**.
Allow Cache-Control Header	**Off**
Loop Detection	**Off**
Follow Redirects	**Off**. 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	**Off**
MIME Back Header Processing	**Off**

Table 19. Advanced settings (continued)

Field	Value
MIME Front Header Processing	**Off**

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 112, Table 16 on page 116.

Table 20. HTTP Basic Authentication properties

Property	Value
Policy Name	`worklight-basicauth`
Order of configured rules	1. `worklight-basicauth_rule_0`: see Table 21 2. `worklight-basicauth_rule_3`: see Table 24 on page 119 3. `worklight-basicauth_rule_1`: see Table 22 4. `worklight-basicauth_rule_2`: see Table 23 on page 119

Table 21. Properties of `worklight-basicauth_rule_0`

Property	Value
Direction	**Client to Server** or **Both Directions**.
Match	• Type = URL • Pattern = `/favicon.ico`
Advanced	"Set Variable" -> `var://service/mpgw/skip-backside = 1`
Result	Not applicable.

Table 22. Properties of `worklight-basicauth_rule_1`

Property	Value
Direction	**Client to Server**.
Match	• Type = URL • Pattern = `*`
AAA	BasicAuth2LTPA • Output: NULL

Table 22. Properties of `worklight-basicauth_rule_1` (continued)

Property	Value
Result	Not applicable.

Table 23. Properties of `worklight-basicauth_rule_2`

Property	Value
Direction	**Server to Client**.
Match	- Type = URL - Pattern = *
Filter	Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see "Sample redirect stylesheet" on page 123. - Output: NULL
Result	Not applicable.

Table 24. Properties of `worklight-basicauth_rule_3`

Property	Value
Direction	**Client to Server**.
Match	- Type = HTTP - HTTP header tag = Cookie - HTTP value match = `*LtpaToken*`
AAA	VerifyLTPA - Output: NULL
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 112, Table 16 on page 116.

Table 25. HTTP Form-Based Login properties

Property	Value
Policy Name	`mpgw-form`
Order of configured rules	1. `mpgw-form_rule_0`: see Table 26 on page 120 2. `mpgw-form_rule_1`: see Table 27 on page 120 3. `mpgw-form_rule_2`: see Table 28 on page 120 4. `mpgw-form_rule_3`: see Table 29 on page 120 5. `mpgw-form_rule_6`: see Table 30 on page 121

Table 26. Properties of `mpgw-form_rule_0`

Property	**Value**
Direction	**Client to Server** or **Both Directions**.
Match	• Type = URL • Pattern = `/favicon.ico`
Advanced	“Set Variable” -> `var://service/mpgw/skip-backside = 1`
Result	Not applicable.

Table 27. Properties of `mpgw-form_rule_1`

Property	**Value**
Direction	**Client to Server**.
Match	• Type = HTTP • HTTP header tag = Cookie • HTTP value match = `*LtpaToken*`
AAA	VerifyLTPA • Output: NULL
Result	Not applicable.

Table 28. Properties of `mpgw-form_rule_2`

Property	**Value**
Direction	**Client to Server**.
Match	• Match with PCRE = on • Type = URL • Pattern = `/(Login|Error)Page\.htm(l)?(\?originalUrl=.*)?`
Transform	Provide a custom stylesheet that builds either a Login or Error HTML page. For a sample stylesheet, see “Sample form login stylesheet” on page 121. **Note:** 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.
Advanced	Select the **set-var** action and specify the service variable: `var://service/routing-url` and value with the endpoint of your login page.
Result	Not applicable.

Table 29. Properties of `mpgw-form_rule_3`

Property	**Value**
Direction	**Client to Server**.
Match	• Type = URL • Pattern = `*`

Table 29. Properties of `mpgw-form_rule_3` (continued)

Property	Value
Advanced	“Convert Query Parameter to XML”. Accept default values for other selections.
AAA	Form2LTPA

Table 30. Properties of `mpgw-form_rule_6`

Property	Value
Direction	**Server to Client**.
Match	- Type = URL - Pattern = *
Filter	Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see “Sample redirect stylesheet” on page 123. - Output: NULL
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 119, Table 28 on page 120.

```
<?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.datapower.com/extensions"
  xmlns:re="http://exslt.org/regular-expressions"
  extension-element-prefixes="dp re"
  exclude-result-prefixes="dp re">
  <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($redirect_uri_preprocess, ';')" />
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="$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" />
                  </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>
            You must 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 redirection and content-type rewriting. You refer to the stylesheet when you create rules to define an HTTP basic authentication policy or an HTML forms-based authentication policy.

You provide a custom stylesheet when you define rule mpgw-form_rule_6 (see "Rules for HTML forms-based authentication" on page 119, Table 30 on page 121), and when you define rule worklight-basicauth_rule_2 (see "Rules for HTTP basic authentication" on page 118, Table 23 on page 119).

```
<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,
              </xsl:when>
              <xsl:otherwise>
                <xsl:value-of select="substring(dp:variable('var://service/error-headers'), 10, 3)
              </xsl:otherwise>
            </xsl:choose>
          </xsl:variable>

          <xsl:choose>
            <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($location, '(\w+):\/\/([^/]+)')">
                  <xsl:if test="position()=3">
                    <xsl:value-of select="." />
                  </xsl:if>
                </xsl:for-each>
              </xsl:variable>
              <xsl:variable name="location_final">
                <xsl:value-of select="re:replace($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-respon
              <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 Worklight adapters and back-end servers by using self-signed certificates

You can configure SSL between IBM Worklight adapters and back-end servers by importing the server's self-signed SSL certificate to the IBM Worklight keystore.

Procedure

1. Check the configuration in the `worklight.properties` file. The configuration might look like this:

```
#################################################################################################
#    Worklight SSL keystore
#################################################################################################
#SSL certificate keystore location.
ssl.keystore.path=conf/default.keystore
#SSL certificate keystore type (jks or PKCS12)
ssl.keystore.type=jks
#SSL certificate keystore password.
ssl.keystore.password=worklight
```

2. Make sure the keystore file exists in the `server/conf` folder of the Worklight project.
3. Export the server's public certificate from the back-end server keystore.

   Note: Export back-end public certificates from the back-end keystore by using `keytool` or `openssl lib`. Do not use the `export` feature using a web browser.
4. Import the back-end server certificate into the Worklight keystore.
5. Restart the Worklight server.

Example

The `CN` name of the back-end certificate should match what is configured in the `adapter.xml` file. For example, consider an `adapter.xml` file that is configured as follows:

```
<protocol>https</protocol>
 <domain>mybackend.com</domain>
```

The back-end certificate should be generated with `CN=mybackend.com`.

As another example, consider the following adapter configuration:

```
<protocol>https</protocol>
<domain>123.124.125.126</domain>
```

The back-end certificate should be generated with `CN=123.124.125.126`.

The following example demonstrates how you complete the configuration by using the keytool program.

1. Create a back-end server keystore with a private certificate for 365 days.

```
keytool -genkey -alias backend -keyalg RSA -validity 365 -keystore backend.keystore -storetype JK
```

   Note that the **First and Last Name** field should be your server URL that you use in adapter configuration XML (for example `mydomain.com` or `localhost`).
2. Configure your back-end server to work with the keystore; for example, in Apache Tomcat, you change the `server.xml` file:

```
<Connector port="443" SSLEnabled="true" maxHttpHeaderSize="8192"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="200"
  enableLookups="false" disableUploadTimeout="true"
  acceptCount="100" scheme="https" secure="true"
  clientAuth="false" sslProtocol="TLS"
  keystoreFile="backend.keystore" keystorePass="password" keystoreType="JKS"
  keyAlias="backend"/>
```

3. Check the connectivity configuration in the adapter XML file:

```
<connectivity>
  <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
    <protocol>https</protocol>
    <domain>mydomain.com</domain>
    <port>443</port>
    <!-- The following properties are used by adapter's key manager for choosing a specific ce
    <sslCertificateAlias></sslCertificateAlias>
    <sslCertificatePassword></sslCertificatePassword>
    -->
  </connectionPolicy>
  <loadConstraints maxConcurrentConnectionsPerNode="2"/>
 </connectivity>
```

4. Export the public certificate from the created back-end server keystore:

```
keytool -export -alias backend -keystore backend.keystore -rfc -file backend.crt
```

5. Import the exported certificate into your Worklight server `default.keystore` file in the `server/conf` directory of the Worklight project:

```
keytool -import -alias backend -file backend.crt -storetype JKS -keystore default.keystore
```

6. Check that the certificate is correctly imported in the keystore:

```
keytool -list -keystore backend.keystore
```

Configuring SSL between 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 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 31. 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 cerficate.crt -reqex
```

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 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 130.

For Android, see "Installing the root CA on Android" on page 133.

Configuring Android

It is important to note that the following flag is set to `true` by default in all 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 134.

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 32. 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: `basicaConstraints = CA:TRUE` For more information, see “Self-signed certificates versus self-signed CAs” on page 126. Ensure that the certificate is in `PEM` format. Ensure that the certificate has a `.crt` file extension.

Table 32. Troubleshoot problems with server certificates (continued)

Problem	Actions to take
Unable to install the root CA on Android. After installation, the certificate does not show up in the system's trusted credentials.	The certificate is an X.509 version 1 certificate or does not have the following certificate extension: `basicConstraints = CA:TRUE` For more information, see "Self-signed certificates versus self-signed CAs" on page 126. Ensure that the certificate is in `PEM` or `DER` format. Ensure that the certificate has a `.crt` file extension.
`"errorCode":"UNRESPONSIVE_HOST","errorMsg":"The service is currently not available."`	This error usually indicates an SSL handshake failure. The client cannot establish trust for the server certificate. 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 127. 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 127. The server certificate is invalid. 1. Check the validity of the server certificate. For more information, see "Tools to use to verify the server certificate" on page 128. 2. Ensure that the CRL URL is valid and reachable. For more information, see "CRL support" on page 128. 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 128.

Table 32. Troubleshoot problems with server certificates (continued)

Problem	Actions to take
SSL works on Android, but does not work on iOS.	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. Worklight sets the mode to `debuggable` in the manifest by default. To make this behavior consistent, set the Android application to `debuggable:false` 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 Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority" on page 125.
After installation, the certificate does not show up in the system's trusted credentials or truststore.	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. For more information about how to properly install the root CA on the device, see the following topics. For iOS, see "Installing the root CA on iOS." For Android, see "Installing the root CA on Android" on page 133.

Related tasks:

"Configuring SSL for Liberty profile" on page 171
Create a keystore, 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`

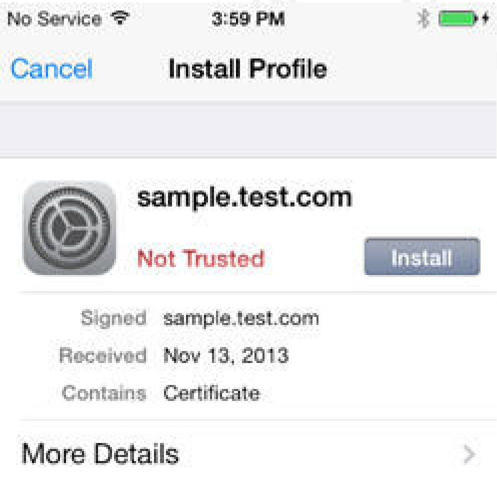
   Note: The following `openssl` flag generates the CA extension:

   ```
   -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.



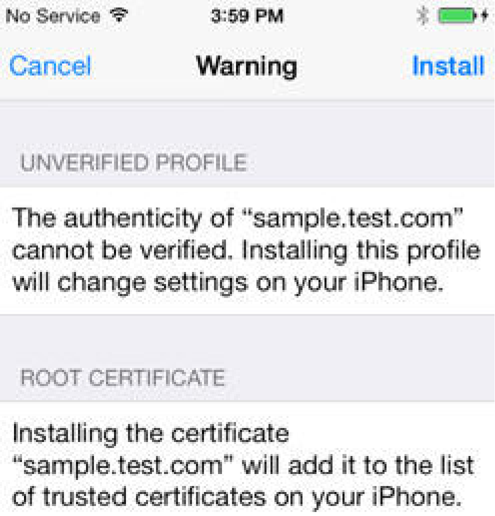
sample.test.com

Not Trusted

Signed sample.test.com

Received Nov 13, 2013

Contains Certificate



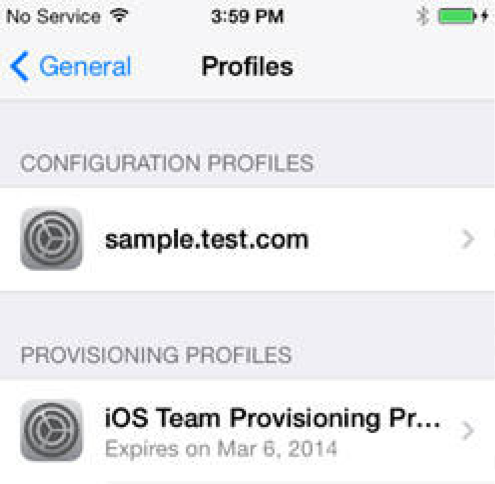
UNVERIFIED PROFILE

The authenticity of “sample.test.com” cannot be verified. Installing this profile will change settings on your iPhone.

ROOT CERTIFICATE

Installing the certificate “sample.test.com” will add it to the list of trusted certificates on your iPhone.

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



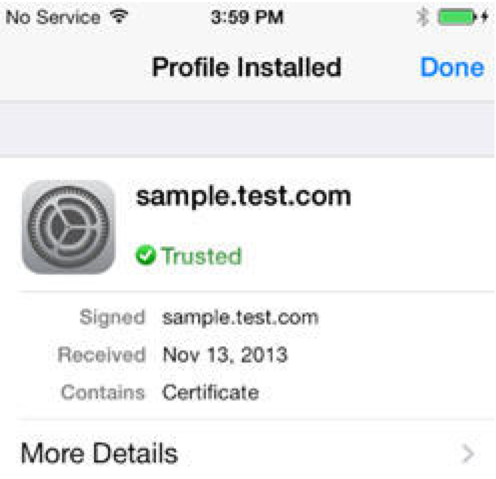
CONFIGURATION PROFILES

sample.test.com

PROVISIONING PROFILES

iOS Team Provisioning Pr...
Expires on Mar 6, 2014

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


No Service 3:59 PM

Profile Installed Done

sample.test.com

Trusted

Signed sample.test.com

Received Nov 13, 2013

Contains Certificate

More Details

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

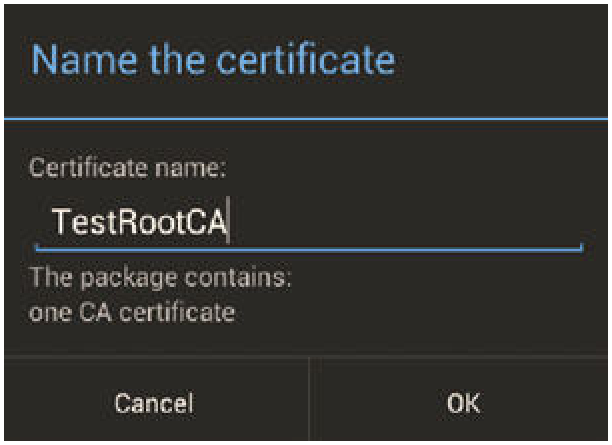
1. Ensure that the root CA is in `PEM` or `DER` file format and has a `.crt` file extension. Convert as needed.
2. Run the following command to view the certificate details.
   ```
   openssl x509 -in certificate.crt -text -noout
   ```
3. Ensure that the certificate is of version X.509 v3. The certificate 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.



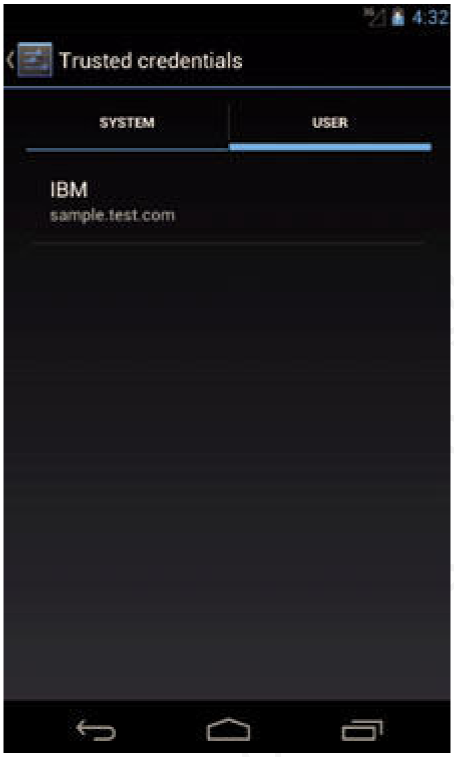
Name the certificate

Certificate name:

TestRootCA

The package contains:
one CA certificate

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



IBM
sample.test.com

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 sta
```

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 that the MySQL wait_timeout system variable to have the connections purged prior to the time that MySQL closes these connections.
d. Click **OK**.

Configuring DB2 HADR seamless failover for Worklight Server and Application Center data sources

You must enable the seamless failover feature with WebSphere Application Server Liberty profile and WebSphere Application Server. With this feature, you can manage an exception when a database fails over and gets rerouted by the DB2 JDBC driver.

By default with DB2 HADR, when the DB2 JDBC driver performs a client reroute after detecting that a database failed over during the first attempt to reuse an existing connection, the driver triggers com.ibm.db2.jcc.am.ClientRerouteException, with ERRORCODE=-4498 and SQLSTATE=08506. WebSphere Application Server maps this exception to com.ibm.websphere.ce.cm.StaleConnectionException before it is received by the application.

In this case, the application would have to catch the exception and execute again the transaction. The Worklight and Application Center runtime environments do not manage the exception but rely on a feature that is called seamless failover. To enable this feature, you must set the **enableSeamlessFailover** JDBC property to "1".

WebSphere Application Server Liberty profile configuration

You must edit the server.xml file, and add the **enableSeamlessFailover** property to the properties.db2.jcc element of the Worklight and Application Center data sources. For example:

```
<dataSource jndiName="jdbc/WorklightAdminDS" transactional="false">
  <jdbcDriver libraryRef="DB2Lib"/>
  <properties.db2.jcc databaseName="WLADMIN"  currentSchema="WLADMSC"
                      serverName="db2server" portNumber="50000"
                      enableSeamlessFailover= "1"
                      user="worklight" password="worklight"/>
</dataSource>
```

WebSphere Application Server configuration

From the WebSphere Application Server administrative console for each Worklight and Application Center data source:

1. Go to **Resources** > **JDBC** > **Data sources** > **DataSource name**.
2. Select **New** and add the following custom property, or update the values if the properties already exist:

   ```
   enableSeamlessFailover : 1
   ```
3. Click **Apply**.
4. Save your configuration.

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 52.

When you install an IBM Worklight edition through IBM Installation Manager, the Application Center is 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 78.

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.

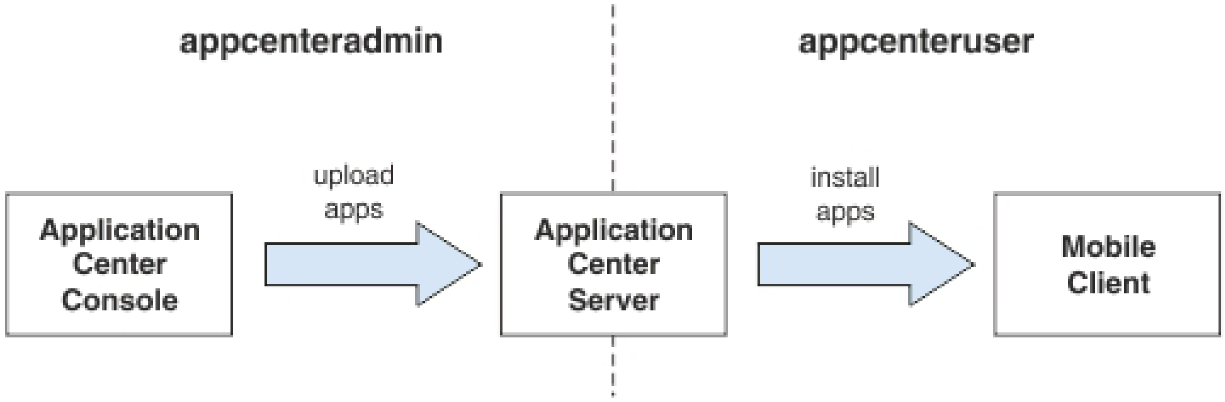


Figure 4. JEE security roles of the Application Center and the components that they influence

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 144 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 852 for how to build the Application Center mobile client.

Related concepts:

“Managing users with LDAP” on page 144
Use the Lightweight Directory Access Protocol (LDAP) registry to manage users.

Related reference:

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.

Configuring WebSphere Application Server full profile

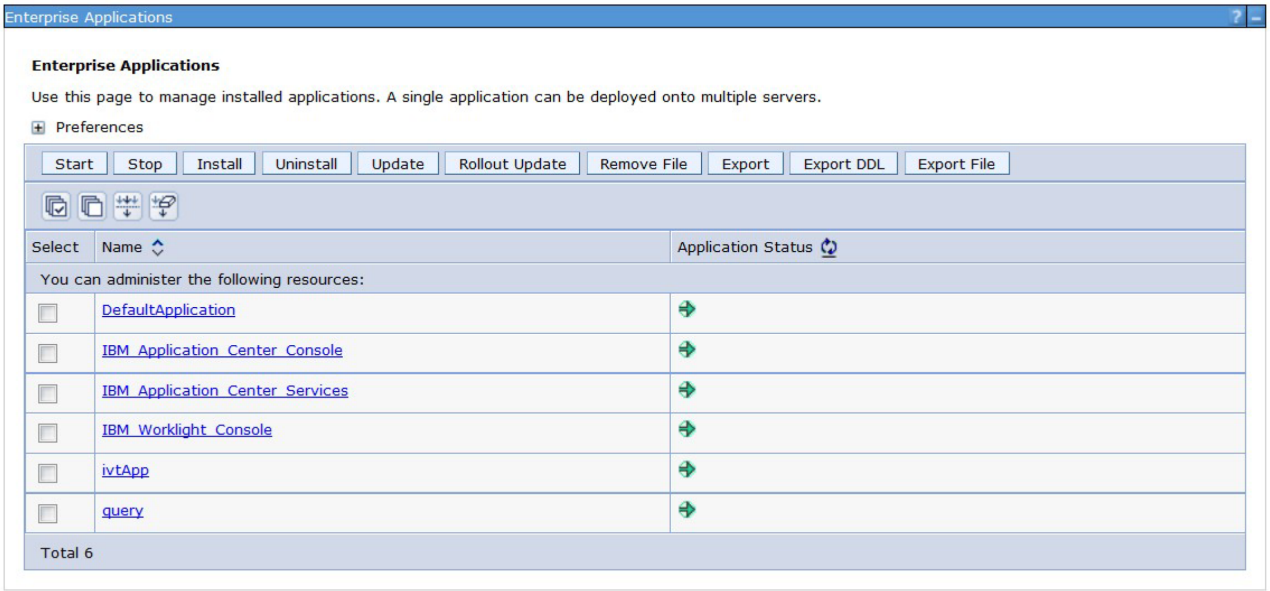
Configure security by mapping the Application Center JEE roles to a set of users for both web applications.

Procedure

You define the basics of user configuration in the WebSphere Application Server console. Access to the console is usually by this address:
`https://localhost:9043/ibm/console/`

1. Select **Security** > **Global Security**.
2. Select **Security Configuration Wizard** to configure users.

   You can manage individual user accounts by selecting **Users and Groups** > **Manage Users**.
3. Map the roles **`appcenteruser`** and **`appcenteradmin`** to a set of users.
   a. Select **Servers** > **Server Types** > **WebSphere application servers**.
   b. Select the server.
   c. In the **Configuration** tab, select **Applications** > **Enterprise applications**.



Enterprise Applications

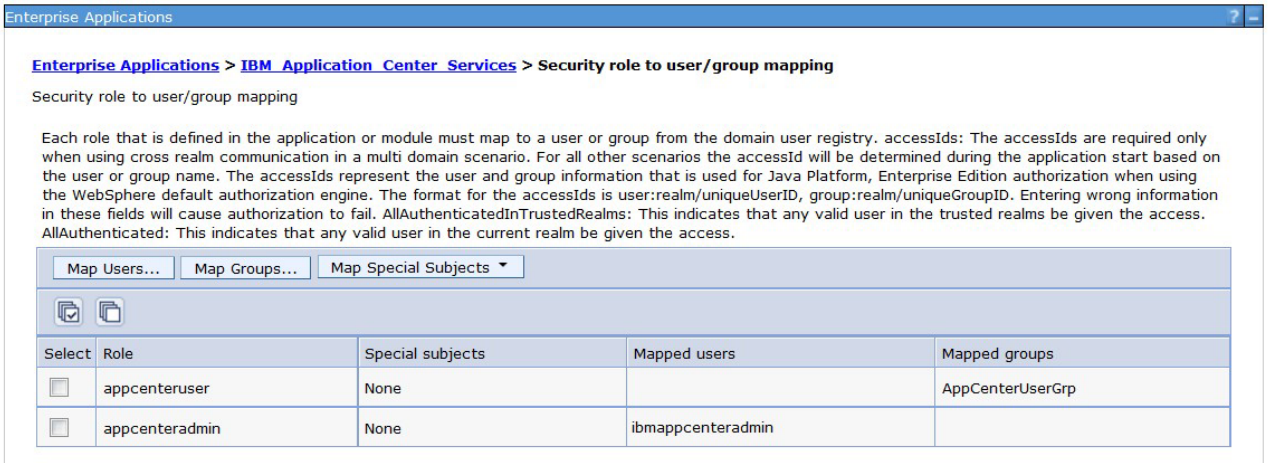
Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Select	Name	Application Status
You can administer the following resources:		
□	DefaultApplication	➔
□	IBM_Application_Center_Console	➔
□	IBM_Application_Center_Services	➔
□	IBM_Worklight_Console	➔
□	ivtApp	➔
□	query	➔
Total 6		

Figure 5. Mapping the Application Center roles

d. Select **IBM_Application_Center_Services**.

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



Enterprise Applications > IBM_Application_Center_Services > Security role to user/group mapping

Security role to user/group mapping

Each role that is defined in the application or module must map to a user or group from the domain user registry. accessIds: The accessIds are required only when using cross realm communication in a multi domain scenario. For all other scenarios the accessId will be determined during the application start based on the user or group name. The accessIds represent the user and group information that is used for Java Platform, Enterprise Edition authorization when using the WebSphere default authorization engine. The format for the accessIds is user:realm/uniqueUserID, group:realm/uniqueGroupID. Entering wrong information in these fields will cause authorization to fail. AllAuthenticatedInTrustedRealms: This indicates that any valid user in the trusted realms be given the access. AllAuthenticated: This indicates that any valid user in the current realm be given the access.

Select	Role	Special subjects	Mapped users	Mapped groups
□	appcenteruser	None		AppCenterUserGrp
□	appcenteradmin	None	ibmappcenteradmin	

Figure 6. 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 **appcenteradmingroup**. These groups are defined through the <basicRegistry> element. You can customize this element or replace it entirely with an <ldapRegistry> element or a <safRegistry> element.

Then, to maintain good response times with a large number of installed applications, for example with 80 applications, you should configure a connection pool for the Application Center database.

Procedure

1. Edit the server.xml file.

   For example:

```
<security-role name="appcenteradmin">
  <group name="appcenteradmingroup"/>
</security-role>
<security-role name="appcenteruser">
  <group name="appcenterusergroup"/>
</security-role>


<basicRegistry id="appcenter">
  <user name="admin" password="admin"/>
  <user name="guest" password="guest"/>
  <user name="demo" password="demo"/>
  <group name="appcenterusergroup">
        <member name="guest" />
        <member name=" demo" />
  </group>
  <group name="appcenteradmingroup">
        <member name="admin" id="admin"/>
  </group>
</basicRegistry>
```

2. Edit the server.xml file to define the **AppCenterPool** size.

```
<connectionManager id="AppCenterPool" minPoolSize="10" maxPoolSize="40"/>
```

3. In the <dataSource> element, define a reference to the connection manager:

```
<dataSource id="APPCNTR" jndiName="jdbc/AppCenterDS" connectionManagerRef="AppCenterPool"
 ...
</dataSource>
```

Configuring Apache Tomcat

You must configure the JEE security roles for the Application Center on the Apache Tomcat web application server.

Procedure

1. In the Apache Tomcat web application server, you configure the roles of **`appcenteruser`** and **`appcenteradmin`** in the `conf/tomcat-users.xml` file. The installation creates the following users:

   ```
   <user username="appcenteradmin" password="admin" roles="appcenteradmin"/>
   <user username="demo" password="demo" roles="appcenteradmin"/>
   <user username="guest" password="guest" roles="appcenteradmin"/>
   ```

2. You can define the set of users as described in the Apache Tomcat documentation, Realm Configuration HOW-TO.

Configuring properties of DB2 JDBC driver in WebSphere Application Server

Add some JDBC custom properties to avoid DB2 exceptions from a WebSphere Application Server that uses the IBM DB2 database.

About this task

When you use WebSphere Application Server with an IBM DB2 database, this exception could occur:

```
Invalid operation: result set is closed. ERRORCODE=-4470, SQLSTATE=null
```

To avoid such exceptions, you must add custom properties in WebSphere Application Server at the Application Center data source level.

Procedure

1. 1. Log in to the WebSphere Application Server administration console.
2. Select **Resources** > **JDBC** > **Data sources** > **Application Center 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**.
7. Click **New**.
8. In the **Name** field, enter **`resultSetHoldability`**.
9. In the Value field, type 1.
10. Change the type to `java.lang.Integer`.
11. Click **OK** and save your changes.

Configuring WebSphere Application Server to support applications in public app stores

Configure WebSphere Application Server full profile and Liberty profile before access to public app stores through application links, because of the use of SSL connections.

The constraint imposed by the use of SSL connections requires the root certificates of public app stores to exist in the WebSphere trust store before you can use application links to access these public stores. The configuration requirement applies to both WebSphere Application Server full profile and Liberty profile.

The root certificate of Google play must be imported into the WebSphere trust store before you can use application links to Google play.

The root certificate of Apple iTunes must be imported into the WebSphere trust store before you can use application links to iTunes.

To use application links to Google play, see "Configuring WebSphere Application Server to support applications in Google play."

To use application links to Apple iTunes, see "Configuring WebSphere Application Server to support applications in Apple iTunes."

Configuring WebSphere Application Server to support applications in Google play

Configure WebSphere Application Server to enable links in the Application Center console to access applications in Google play.

About this task

Follow this procedure to import the root certificate of Google play into the WebSphere trust store. You must import this certificate before the Application Center can support links to applications stored in Google Play.

Procedure

1. Log in to the WebSphere Application Server console and navigate to **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates**.
2. Click **Retrieve from port**.
3. In the **Host** field, enter `play.google.com`.
4. In the **Port** field, enter `443`.
5. In the **Alias** field, enter `play.google.com`.
6. Click **Retrieve signer information**.
7. Click **OK** and save the configuration.

Configuring WebSphere Application Server to support applications in Apple iTunes

Configure WebSphere Application Server to enable links in the Application Center console to access applications in Apple iTunes.

About this task

Follow this procedure to import the root certificate of Apple iTunes into the WebSphere trust store. You must import this certificate before the Application Center can support links to applications stored in iTunes.

Procedure

1. Log in to the WebSphere Application Server console and navigate to **Security** > **SSL certificate and key management** > **Key stores and certificates** > **NodeDefaultTrustStore** > **Signer certificates**.
2. Click **Retrieve from port**.
3. In the **Host** field, enter `itunes.apple.com`.
4. In the **Port** field, enter `443`.
5. In the **Alias** field, enter `itunes.apple.com`.
6. Click **Retrieve signer information**.
7. Click **OK** and save the configuration.

Configuring Liberty Profile when IBM JDK is used

Configure Liberty Profile to use default JSSE socket factories instead of SSL socket factories of WebSphere Application Server when IBM JDK is used.

Purpose

This configuration is required only when IBM JDK is used. (The configuration does not apply for use of Oracle JDK.) By default, IBM JDK uses the SSL socket factories of WebSphere Application Server. These factories are not supported by Liberty Profile, so you must edit the `java.security` file from the JDK.

- You must uncomment the two lines that set the SSL socket factories to the default JSSE factories.
- You must comment out the two lines that set the SSL socket factories to the SSL socket factories of WebSphere Application Server.

Exception when WebSphere Application Server SSL socket factories are used

If you use the SSL socket factories of WebSphere Application Server, this exception occurs when the connection with SSL is attempted. In this case, you must edit the `java.security` file.

```
java.net.SocketException: java.lang.ClassNotFoundException: Cannot find the specified class com.ibm.w
      at javax.net.ssl.DefaultSSLSocketFactory.a(SSLSocketFactory.java:11)
      at javax.net.ssl.DefaultSSLSocketFactory.createSocket(SSLSocketFactory.java:6)
      at com.ibm.net.ssl.www2.protocol.https.c.afterConnect(c.java:161)
      at com.ibm.net.ssl.www2.protocol.https.d.connect(d.java:36)
      at sun.net.www.protocol.http.HttpURLConnection.getInputStream(HttpURLConnection.java:1184)
      at java.net.HttpURLConnection.getResponseCode(HttpURLConnection.java:390)
      at com.ibm.net.ssl.www2.protocol.https.b.getResponseCode(b.java:75)
      at com.ibm.ws.jmx.connector.client.rest.internal.RESTMBeanServerConnection.loadJMXServerInfo(RE
      at com.ibm.ws.jmx.connector.client.rest.internal.RESTMBeanServerConnection.<init>(RESTMBeanServ
      at com.ibm.ws.jmx.connector.client.rest.internal.Connector.connect(Connector.java:315)
      at com.ibm.ws.jmx.connector.client.rest.internal.Connector.connect(Connector.java:103)
```

Edited content of the java.security file

This sample shows the edits made to the `java.security` file.

```
# Default JSSE socket factories
ssl.SocketFactory.provider=com.ibm.jsse2.SSLSocketFactoryImpl
ssl.ServerSocketFactory.provider=com.ibm.jsse2.SSLServerSocketFactoryImpl
# WebSphere socket factories (in cryptosf.jar)
#ssl.SocketFactory.provider=com.ibm.websphere.ssl.protocol.SSLSocketFactory
#ssl.ServerSocketFactory.provider=com.ibm.websphere.ssl.protocol.SSLServerSocketFactory
```

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 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`.
   - **ibm.appcenter.ldap.cache.expiration.seconds**: delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours.

     Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **ibm.appcenter.ldap.cache.expiration.seconds**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data:

     ```
     acdeploytool.sh -clearLdapCache -s serverurl -c context -u user -p password
     ```

     See Using the stand-alone tool to clear the LDAP cache for details.

   a. Enter the value of each property.

   b. Click **OK** and save the configuration.

6. **Option:** *If the LDAP external SASL authentication mechanism is required to bind to the LDAP server,* configure the **ibm.appcenter.ldap.active.sasl** property, which defines the value of the security authentication mechanism. The value depends on the LDAP server; usually, it is set to “EXTERNAL”.
7. **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.

8. **Option:** *If LDAP referrals must be handled, follow this step.* Configure **ibm.appcenter.ldap.referral**: support of referrals by the JNDI API. • If no value is given, the JNDI API will not handle LDAP referrals. Possible values are:
   - **ignore**: ignores referrals found in the LDAP server.
   - **follow**: automatically follows any referrals found in the LDAP server.
   - **throw**: causes an exception to occur for each referral found in the LDAP server.

   a. Enter the value of the property and click **OK**.

b. Save the configuration.

9. **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.

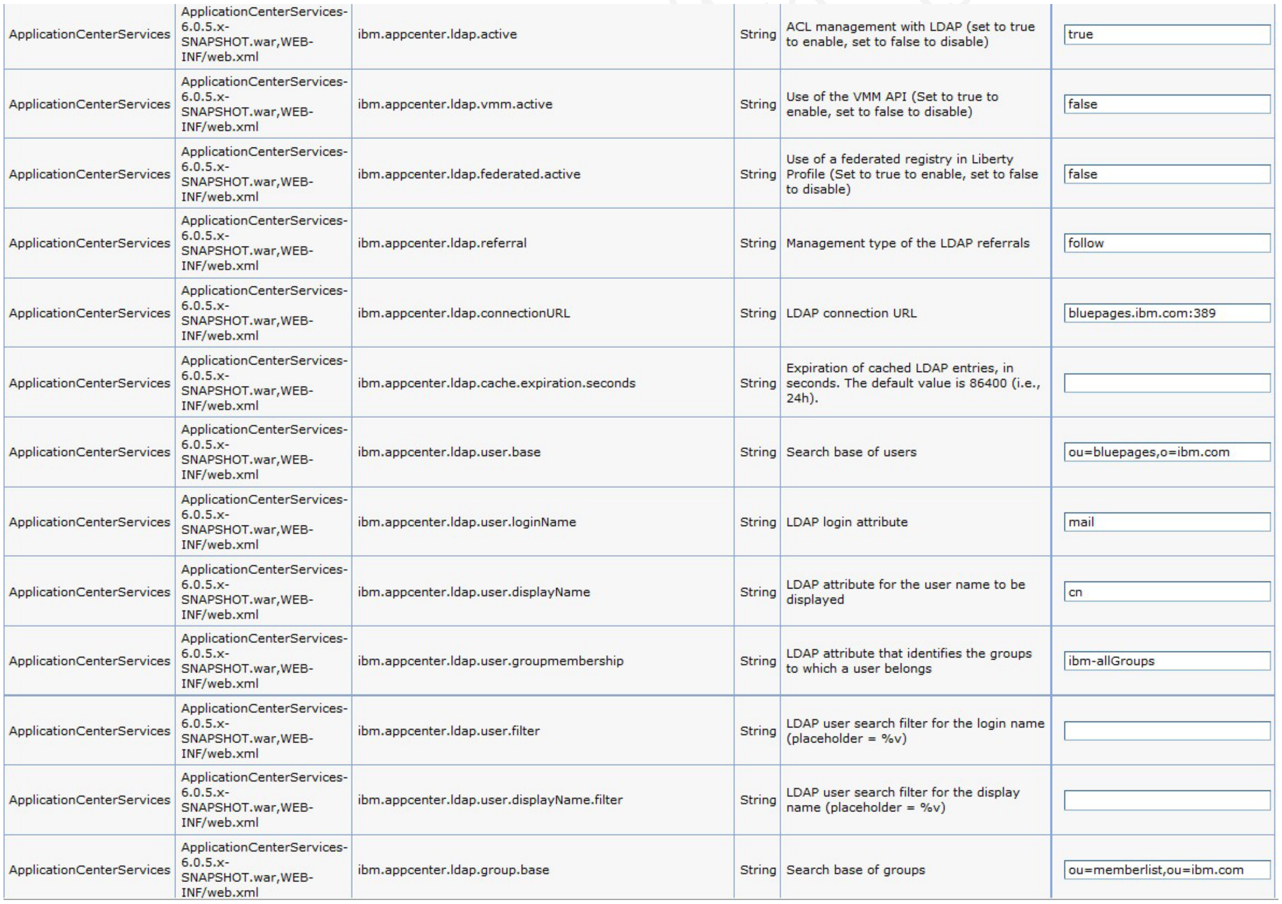
   a. Enter the value of each optional property and click **OK**.

   b. Save the configuration.

Results

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

See "List of JNDI properties for the Application Center" on page 172 for a complete list of properties that you can set.



ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.active	String	ACL management with LDAP (set to true to enable, set to false to disable)	true
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.vmm.active	String	Use of the VMM API (Set to true to enable, set to false to disable)	false
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.federated.active	String	Use of a federated registry in Liberty Profile (Set to true to enable, set to false to disable)	false
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.referral	String	Management type of the LDAP referrals	follow
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.connectionURL	String	LDAP connection URL	bluepages.ibm.com:389
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.cache.expiration.seconds	String	Expiration of cached LDAP entries, in seconds. The default value is 86400 (i.e., 24h).	
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.base	String	Search base of users	ou=bluepages,o=ibm.com
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.loginName	String	LDAP login attribute	mail
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.displayName	String	LDAP attribute for the user name to be displayed	cn
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.groupmembership	String	LDAP attribute that identifies the groups to which a user belongs	ibm-allGroups
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.filter	String	LDAP user search filter for the login name (placeholder = %v)	
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.user.displayName.filter	String	LDAP user search filter for the display name (placeholder = %v)	
ApplicationCenterServices	ApplicationCenterServices-6.0.5.x-SNAPSHOT.war,WEB-INF/web.xml	ibm.appcenter.ldap.group.base	String	Search base of groups	ou=memberlist,ou=ibm.com

Figure 7. Environment entries and their values (LDAP and WebSphere Application Server V7)

Note: The attribute **ibm.appcenter.ldap.user.groupmembership** specifies the groups of a member when you use LDAP without VMM. This property is the inverse of **ibm.appcenter.ldap.group.uniquemember**. This property is optional, but if it is specified, the LDAP access is faster. One or both of the properties **ibm.appcenter.ldap.user.groupmembership** or **ibm.appcenter.ldap.group.uniquemember** must be specified. The system behaves faster if both are specified, but some LDAP implementations do not support the group membership attribute for a user. In this case, if **ibm.appcenter.ldap.group.uniquemember** is specified, the property **ibm.appcenter.ldap.user.groupmembership** is not set.

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 WebShere 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.
   - `ibm.appcenter.ldap.cache.expiration.seconds = `: the delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours.

     Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **`ibm.appcenter.ldap.cache.expiration.seconds`**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data:

     ```
     acdeploytool.sh -clearLdapCache -s serverurl -c context -u user -p password
     ```

     See Using the stand-alone tool to clear the LDAP cache for details.

   See “List of JNDI properties for the Application Center” on page 172 for a complete list of properties that you can set.

   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 on page 151 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.

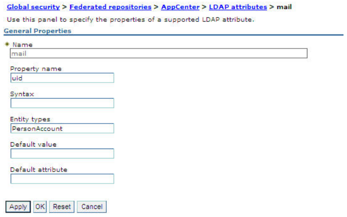
About this task

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**.



Global security > Federated repositories > AppCenter > LDAP attributes > mail

Use this panel to specify the properties of a supported LDAP attribute.

General Properties

* Name: mail

Property name: uid

Syntax:

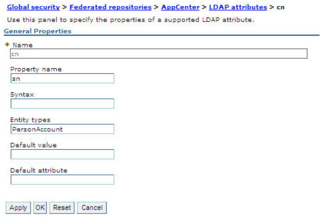
Entity types: PersonAccount

Default value:

Default attribute:

Figure 8. 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**.



Global security > Federated repositories > AppCenter > LDAP attributes > cn

Use this panel to specify the properties of a supported LDAP attribute.

General Properties

* Name: cn

Property name: sn

Syntax:

Entity types: PersonAccount

Default value:

Default attribute:

Figure 9. 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. Enter the required property-value pairs in the form. To enter each pair, click **New**, enter the property and its value, and click **OK**.

       Property-value pairs:
       - `ibm.appcenter.ldap.vmm.active = true`
       - `ibm.appcenter.ldap.active = true`
       - `ibm.appcenter.ldap.cache.expiration.seconds =` *`delay_in_seconds`*

         Enter the delay in seconds before the LDAP cache expires. If you do not enter a value, the default value is 86400, which is equal to 24 hours.

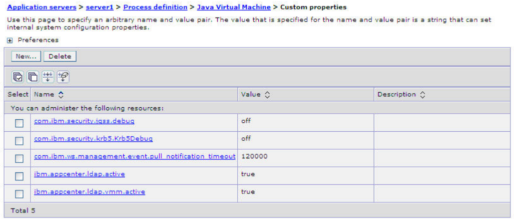
Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **ibm.appcenter.ldap.cache.expiration.seconds**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data:

```
acdeploytool.sh -clearLdapCache -s serverurl -c context -u user -p password
```

See Using the stand-alone tool to clear the LDAP cache for details.

Results

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



Application servers > server1 > Process definition > Java Virtual Machine > Custom properties

Use this page to specify an arbitrary name and value pair. The value that is specified for the name and value pair is a string that can set internal system configuration properties.

Select	Name	Value	Description
You can administer the following resources:			
□	com.ibm.security.jgss.debug	off	
□	com.ibm.security.krb5.Krb5Debug	off	
□	com.ibm.ws.management.event.pull_notification_timeout	120000	
□	ibm.appcenter.ldap.active	true	
□	ibm.appcenter.ldap.vmm.active	true	
Total 5			

Figure 10. 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 one or more LDAP registries in the `server.xml` file and you 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 or, since WebSphere Application Server Liberty Profile V8.5.5, a federated registry that uses several LDAP registries. 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 one or several LDAP registry definitions 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="(&amp;(cn=%v)(|(objectclass=groupOfNames)(objectclass=groupOfUniqueNames)))
         userFilter="(&amp;(emailAddress=%v)(objectclass=ibmPerson))"
         groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember"
         userIdMap="*:emailAddress"/>
   </ldapRegistry>
   ```

   For information about the parameters used in this example, see the WebSphere Application Server V8.5 user documentation.
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 33. JNDI properties for configuring ACL management with LDAP in the server.xml file

Property	Description
ibm.appcenter.ldap.active	Set to true to enable LDAP; set to false to disable LDAP.

Table 33. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
ibm.appcenter.ldap.federated.active	Since WebSphere Application Server Liberty Profile V8.5.5: set to `true` to enable use of the federated registry; set to `false` to disable use of the federated registry, which is the default setting.
ibm.appcenter.ldap.connectionURL	LDAP connection URL.
ibm.appcenter.ldap.user.base	Search base of users.
ibm.appcenter.ldap.user.loginName	LDAP login attribute.
ibm.appcenter.ldap.user.displayName	LDAP attribute for the user name to be displayed, for example, a person's full name.
ibm.appcenter.ldap.group.base	Search base of groups.
ibm.appcenter.ldap.group.name	LDAP attribute for the group name.
ibm.appcenter.ldap.group.uniquemember	LDAP attribute that identifies the members of a group.
ibm.appcenter.ldap.user.groupmembership	LDAP attribute that identifies the groups to which a user belongs. Specifies the groups of a member when you use LDAP without VMM. This property is the inverse of ibm.appcenter.ldap.group.uniquemember. This property is optional, but if it is specified, the LDAP access is faster.
ibm.appcenter.ldap.group.nesting	Management of nested groups: if nested groups are not managed, set the value to `false`.
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. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.displayName.filter	LDAP user search filter for the attribute of user display name. Use `%v` as the placeholder for the display name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.

Table 33. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
ibm.appcenter.ldap.group.filter	LDAP group search filter. Use %v as the placeholder for the group attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.active.sasl	The value of the security authentication mechanism when the LDAP external SASL authentication mechanism is required to bind to the LDAP server. The value depends on the LDAP server; usually, it is set to "EXTERNAL".
ibm.appcenter.ldap.security.binddn	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.
ibm.appcenter.ldap.security.bindpwd	Property that identifies the password of the user permitted to search the LDAP directory. Use this property only if security binding is required. The password can be encoded with the "Liberty Profile securityUtility" tool. Run the tool and then set the value of this property to the encoded password generated by the tool. The supported encoding types are xor and aes. Edit the Liberty Profile `server.xml` file to check whether the *classloader* is enabled to load the JAR file that decodes the password.
ibm.appcenter.ldap.cache.expiration.seconds	Delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours. Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **ibm.appcenter.ldap.cache.expiration.seconds**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data: `acdeploytool.sh -clearLdapCache -s serverurl -c context -u` See Using the stand-alone tool to clear the LDAP cache for details.

Table 33. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
ibm.appcenter.ldap.referral	Property that indicates whether referrals are supported by the JNDI API. If no value is given, the JNDI API will not handle LDAP referrals. Possible values are: • `ignore`: ignores referrals found in the LDAP server. • `follow`: automatically follows any referrals found in the LDAP server. • `throw`: causes an exception to occur for each referral found in the LDAP server.

See "List of JNDI properties for the Application Center" on page 172 for a complete list of properties that you can set.

Note: One or both of the properties **ibm.appcenter.ldap.user.groupmembership** or **ibm.appcenter.ldap.group.uniquemember** must be specified. The system behaves faster if both are specified, but some LDAP implementations do not support the group membership attribute for a user. In this case, if **ibm.appcenter.ldap.group.uniquemember** is specified, the property **ibm.appcenter.ldap.user.groupmembership** is not set.

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:636"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.loginName" value="uid"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.base" value="dc=ibm,dc=com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.group.base" value="dc=ibm,dc=com"/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName" value="sn"/>
<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"/>
<jndiEntry jndiName="ibm.appcenter.ldap.cache.expiration.seconds" value=43200"/>
<jndiEntry jndiName="ibm.appcenter.ldap.security.sasl" value='"EXTERNAL"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.referral" value='"follow"'/>
<jndiEntry jndiName="ibm.appcenter.ldap.user.filter" value='"(&amp;(uid=%v)(objectclass=inetOrgPer
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName.filter" value='"(&amp;(cn=%v)(objectclass
<jndiEntry jndiName="ibm.appcenter.ldap.group.filter" value='"(&amp;(cn=%v)(|(objectclass=groupOfN
```

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"/>
 ...
</Host>
```

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. 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 Services web application (applicationcenter.war) and of the Application Center Console web application (appcenterconsole.war) 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>
            <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="fals
```

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 34. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat

Property	Description
ibm.appcenter.ldap.active	Set to `true` to enable LDAP; set to `false` to disable LDAP.
ibm.appcenter.ldap.connectionURL	LDAP connection URL.
ibm.appcenter.ldap.user.base	Search base of users.
ibm.appcenter.ldap.user.loginName	LDAP login attribute.
ibm.appcenter.ldap.user.displayName	LDAP attribute for the user name to be displayed, for example, a person's full name.
ibm.appcenter.ldap.group.base	Search base of groups.
ibm.appcenter.ldap.group.name	LDAP attribute for the group name.
ibm.appcenter.ldap.group.uniquemember	LDAP attribute that identifies the members of a group.
ibm.appcenter.ldap.user.groupmembership	LDAP attribute that identifies the groups to which a user belongs. Specifies the groups of a member when you use LDAP without VMM. This property is the inverse of ibm.appcenter.ldap.group.uniquemember. This property is optional, but if it is specified, the LDAP access is faster.
ibm.appcenter.ldap.group.nesting	Management of nested groups: if nested groups are not managed, set the value to `false`.
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. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.displayName.filter	LDAP user search filter for the attribute of user display name. Use `%v` as the placeholder for the display name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.group.filter	LDAP group search filter. Use `%v` as the placeholder for the group attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.

Table 34. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat (continued)

Property	Description
ibm.appcenter.ldap.active.sasl	The value of the security authentication mechanism when the LDAP external SASL authentication mechanism is required to bind to the LDAP server. The value depends on the LDAP server; usually, it is set to "EXTERNAL".
ibm.appcenter.ldap.security.binddn	Property that identifies the distinguished name of the user permitted to search the LDAP directory. Use this property only if security binding is required.
ibm.appcenter.ldap.security.bindpwd	Property that identifies the password of the user permitted to search the LDAP directory. Use this property only if security binding is required.
ibm.appcenter.ldap.cache.expiration.seconds	Delay in seconds before the LDAP cache expires. If no value is entered, the default value is 86400, which is equal to 24 hours. Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **ibm.appcenter.ldap.cache.expiration.seconds**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call this command to clear the cache of LDAP data: `acdeploytool.sh -clearLdapCache -s serverurl -c context -u` See Using the stand-alone tool to clear the LDAP cache for details.
ibm.appcenter.ldap.referral	Property that indicates whether referrals are supported by the JNDI API. If no value is given, the JNDI API will not handle LDAP referrals. Possible values are: • `ignore`: ignores referrals found in the LDAP server. • `follow`: automatically follows any referrals found in the LDAP server. • `throw`: causes an exception to occur for each referral found in the LDAP server.

The example shows properties defined in the `server.xml` file.

See “List of JNDI properties for the Application Center” on page 172 for a complete list of properties that you can set.

Note: One or both of the properties **ibm.appcenter.ldap.user.groupmembership** or **ibm.appcenter.ldap.group.uniquemember** must be specified. The system behaves faster if both are specified, but some LDAP implementations do not support the group membership attribute for a user. In this case, if **ibm.appcenter.ldap.group.uniquemember** is specified, the property **ibm.appcenter.ldap.user.groupmembership** is not set.

```
<Environment name="ibm.appcenter.ldap.active" value="true" type="java.lang.String" override="false
<Environment name="ibm.appcenter.ldap.connectionURL" value="ldaps://employees.com:636" type="java.
<Environment name="ibm.appcenter.ldap.user.base" value="dc=ibm,dc=com" type="java.lang.String" ove
<Environment name="ibm.appcenter.ldap.user.loginName" value="uid" type="java.lang.String" override
<Environment name="ibm.appcenter.ldap.user.displayName" value="cn" type="java.lang.String" overric
<Environment name="ibm.appcenter.ldap.user.groupmembership" value="ibm-allGroups" type="java.lang.
<Environment name="ibm.appcenter.ldap.group.base" value="dc=ibm,dc=com" type="java.lang.String" ov
<Environment name="ibm.appcenter.ldap.group.name" value="cn" type="java.lang.String" override="fal
<Environment name="ibm.appcenter.ldap.group.uniquemember" value="uniquemember" type="java.lang.Str
<Environment name="ibm.appcenter.ldap.cache.expiration.seconds" value="43200" type="java.lang.Stri
<Environment name="ibm.appcenter.ldap.security.sasl" value="EXTERNAL" type="java.lang.String" over
<Environment name="ibm.appcenter.ldap.security.referral" value="follow" type="java.lang.String" ov
<Environment name="ibm.appcenter.ldap.user.filter" value="(&amp;(uid=%v)(objectclass=inetOrgPerson
<Environment name="ibm.appcenter.ldap.user.displayName.filter" value="(&amp;(cn=%v)(objectclass=ir
<Environment name="ibm.appcenter.ldap.group.filter" value="(&amp;(cn=%v)(|(objectclass=groupOfName
```

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**).

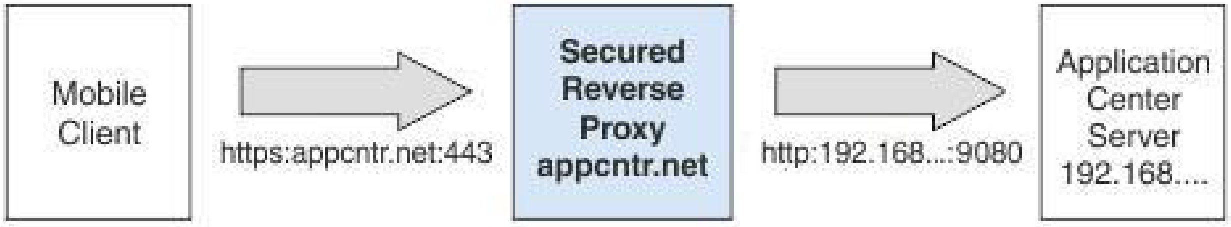

Mobile Client
https:appcntr.net:443
Secured Reverse Proxy appcntr.net
http:192.168...:9080
Application Center Server 192.168....

Figure 11. Configuration with secured reverse proxy

Table 35. The endpoint properties

Property name	Purpose	Example
ibm.appcenter.services.endpoint	This property enables the Application Center console to locate the Application Center REST services. The value of this property must be specified as the external address and context root of the applicationcenter.war web application. You can use the asterisk (*) character as wildcard to specify that the Application Center REST services use the same value as the Application Center console. For example: *://*:*/appcenter means use the same protocol, host, and port as the Application Center console, but use appcenter as context root. This property must be specified for the Application Center console application.	https://appcntr.net:443/applicationcenter
ibm.appcenter.proxy.protocol	This property specifies the protocol required for external applications to connect to the Application Center.	https
ibm.appcenter.proxy.host	This property specifies the host name required for external applications to connect to the Application Center.	**appcntr.net**
ibm.appcenter.proxy.port	This property specifies the port required for external applications to connect to the Application Center.	443

See "List of JNDI properties for the Application Center" on page 172 for a complete list of properties that you can set.

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 Application Center services and the Application Center console applications.

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.proxy.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.
   - You can use the asterisk (*) character as wildcard to specify that the Application Center REST services use the same value as the Application Center console. For example: *://*:*/appcenter means use the same protocol, host, and port as the Application Center console, but use appcenter as context root.
10. Click **OK** and save the configuration. For a complete list of JNDI properties that you can set, see “List of JNDI properties for the Application Center” on page 172.

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 36. Properties in the server.xml file for configuring the endpoint of the application resources

Property	Description
ibm.appcenter.services.endpoint	The URI of the Application Center REST services (applicationcenter.war). In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI and not the internal URI inside the local LAN.
ibm.appcenter.proxy.protocol	The protocol of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.
ibm.appcenter.proxy.host	The hostname of the application resources URI.
ibm.appcenter.proxy.port	The port of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.

See "List of JNDI properties for the Application Center" on page 172 for a complete list of JNDI properties that you can set.

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/applicationcent
<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"/>
```

For **ibm.appcenter.services.endpoint**, you can use the asterisk (*) character as wildcard to specify that the Application Center services use the same value as the Application Center console. For example: *://*:*/appcenter means use the same protocol, host, and port as the Application Center console, but use appcenter as context root.

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 in the <context> section of the corresponding 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 37. Properties in the server.xml file for configuring the endpoint of the application resources

Property	Type	Description
ibm.appcenter.services.endpoint	java.lang.String	The URI of the Application Center REST services (applicationcenter.war). In a scenario with a firewall or a secured reverse proxy, this URI must be the external URI and not the internal URI inside the local LAN.
ibm.appcenter.proxy.protocol	java.lang.String	The protocol of the application resources URI. This property is optional. It is only needed if the protocol of the external and of the internal URI are different.
ibm.appcenter.proxy.host	java.lang.String	The hostname of the application resources URI.
ibm.appcenter.proxy.port	java.lang.Integer	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.

See "List of JNDI properties for the Application Center" on page 172 for a complete list of JNDI properties that you can set.

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.

In the context section of the Application Center Console application:

```
<Environment name="ibm.appcenter.services.endpoint" value="https://appcntr.net:443/applicationcent
```

You can use the asterisk (*) character as wildcard to specify that the Application Center services use the same value as the Application Center console. For example: *://*:*/appcenter means use the same protocol, host, and port as the Application Center console, but use appcenter as context root.

In the context section of the Application Center Services application:

```
<Environment name="ibm.appcenter.proxy.protocol" value="https" type="java.lang.String" override="fals
<Environment name="ibm.appcenter.proxy.host" value="appcntr.net" type="java.lang.String" override="fa
<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 keystore, 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 keystore 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 keystore 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 keystore service object entry to the `server.xml` file. The **`keyStore`** element is called **`defaultKeyStore`** and contains the keystore 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/...`

List of JNDI properties for the Application Center

Here is a list of the JNDI properties that can be configured for the Application Center.

Table 38. List of the JNDI properties for the Application Center

Property	Description
`appcenter.database.type`	The database type, which is only required when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.jndi.name`	The JNDI name of the database. This parameter is the normal mechanism to specify the database. The default value is `java:comp/env/jdbc/AppCenterDS`.
`appcenter.openjpa.ConnectionDriverName`	The fully qualified class name of the database connection driver class. This property is only needed when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionPassword`	The password for the database connection. This property is only needed when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionURL`	The URL specific to the database connection driver class. This property is only needed when the database is not specified in **`appcenter.jndi.name`**.
`appcenter.openjpa.ConnectionUserName`	The user name or the database connection. This property is only needed when the database is not specified in **`appcenter.jndi.name`**.
`ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate`	Specifies whether the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. Set to `true` to enable or `false` to disable. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 863.

Table 38. List of the JNDI properties for the Application Center (continued)

Property	Description
ibm.appcenter.apns.p12.certificate.location	The path to the file of the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. For example, /Users/someUser/someDirectory/apache-tomcat/conf/AppCenter_apns_dev_cert.p12. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 863.
ibm.appcenter.apns.p12.certificate.password	The password of the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 863.
ibm.appcenter.forceUpgradeDBTo60	The database design was changed starting from Worklight version 6.0. The database is automatically updated when the Application Center web application starts. If you want to repeat this update, you can set this parameter to true and start the web application again. Later you can set this parameter to false.
ibm.appcenter.gcm.signature.googleapikey	The Google API key that enables the Application Center to send push notifications about updates for Android applications. For example, AIxaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs. See "Configuring the Application Center server for connection to Google Cloud Messaging" on page 861.
ibm.appcenter.ios.plist.onetimeurl	Specifies whether URLs stored in iOS plist manifests use the one-time URL mechanism without credentials. If you set this property to true, the security level is medium since the one-time URLs are generated with a cryptographic mechanism so that nobody can guess the URL. However, they do not require the user to log in when you use these URLs. Setting this property to false is maximally secure, since the user is then required to log in for each URL. However, requesting the user to log in multiple times when you install an iOS application can degrade the user experience. See "Installing the client on an iOS mobile device" on page 899.
ibm.appcenter.ldap.active	Specifies whether Application Center is configured for LDAP. Set to true to enable LDAP; set to false to disable LDAP. See "Managing users with LDAP" on page 144.
ibm.appcenter.ldap.active.sasl	Specifies the security authentication mechanism when the LDAP external SASL authentication mechanism is required to bind to the LDAP server. The value depends on the LDAP server and it is typically set to EXTERNAL.

Table 38. List of the JNDI properties for the Application Center (continued)

Property	Description
ibm.appcenter.ldap.cache.expiration.seconds	The Application Center maintains a cache of LDAP data and the changes become visible only after the cache expires. Specify the amount of time in seconds an entry in the LDAP cache is valid. Set this property to a value larger than 3600 (1 hour) to reduce the amount of LDAP requests. If no value is entered, the default value is 86400, which is equal to 24 hours. If you need to manually clear the cache of LDAP data, enter this command: `acdeploytool.sh -clearLdapCache -s serverurl -c context -u user -p` See Using the stand-alone tool to clear the LDAP cache for details.
ibm.appcenter.ldap.connectionURL	The URL to access the LDAP server when no VMM is used. See “Configuring LDAP ACL management (Liberty Profile)” on page 156 and “Configuring LDAP ACL management (Apache Tomcat)” on page 162.
ibm.appcenter.ldap.federated.active	Specifies whether Application Center is configured for LDAP with federated repositories. Since WebSphere Application Server Liberty Profile V8.5.5. set this property to `true` to enable use of the federated registry. Set this property to `false` to disable use of the federated registry, which is the default setting. See “Managing users with LDAP” on page 144.
ibm.appcenter.ldap.group.base	The search base to find groups when you use LDAP without VMM. See “Configuring LDAP ACL management (Liberty Profile)” on page 156 and “Configuring LDAP ACL management (Apache Tomcat)” on page 162.
ibm.appcenter.ldap.group.filter	LDAP group search filter. Use `%v` as the placeholder for the group attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.group.name	The group name attribute when you use LDAP without VMM. See “Configuring LDAP ACL management (Liberty Profile)” on page 156 and “Configuring LDAP ACL management (Apache Tomcat)” on page 162.
ibm.appcenter.ldap.group.nesting	Specifies whether the LDAP contains nested groups (that is, groups in groups) when you use LDAP without VMM. Setting this property to `false` speeds up the LDAP access since the groups are then not searched recursively. See “Configuring LDAP ACL management (Liberty Profile)” on page 156 and “Configuring LDAP ACL management (Apache Tomcat)” on page 162.

Table 38. List of the JNDI properties for the Application Center (continued)

Property	Description
`ibm.appcenter.ldap.group.uniquemember`	Specifies the members of a group when you use LDAP without VMM. This property is the inverse of **`ibm.appcenter.ldap.user.groupmembership`**. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
`ibm.appcenter.ldap.referral`	Specifies whether referrals are supported by the JNDI API. If no value is given, the JNDI API does not handle LDAP referrals. Here are the possible values: • `ignore`: ignores referrals that are found in the LDAP server. • `follow`: automatically follows any referrals that are found in the LDAP server. • `throw`: causes an exception to occur for each referral found in the LDAP server.
`ibm.appcenter.ldap.security.binddn`	The distinguished name of the user that is allowed to search the LDAP directory. Use this property only if security binding is required. The password can be encoded with the "Liberty Profile securityUtility" tool. Run the tool and then set the value of this property to the encoded password generated by the tool. The supported encoding types are `xor` and `aes`. Edit the Liberty Profile `server.xml` file to check whether the *classloader* is enabled to load the JAR file that decodes the password.See "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
`ibm.appcenter.ldap.security.bindpwd`	The password of the user that is permitted to search the LDAP directory. Use this property only if security binding is required. See "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
`ibm.appcenter.ldap.security.sasl`	If set, security authentication is used when you connect to LDAP without VMM. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
`ibm.appcenter.ldap.user.base`	The search base to find users when you use LDAP without VMM. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
`ibm.appcenter.ldap.user.displayName`	The display name attribute, such as the user's real name, when you use LDAP without VMM. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.

Table 38. List of the JNDI properties for the Application Center (continued)

Property	Description
ibm.appcenter.ldap.displayName.filter	LDAP user search filter for the attribute of **ibm.appcenter.ldap.user.displayName**. Use %v as the placeholder for the display name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.user.filter	LDAP user search filter for the attribute of **ibm.appcenter.ldap.user.loginName**. Use %v as the placeholder for the login name attribute. This property is only required when LDAP users and groups are defined in the same subtree; that is, when the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value.
ibm.appcenter.ldap.user.groupmembership	Specifies the groups of a member when you use LDAP without VMM. This property is the inverse of **ibm.appcenter.ldap.group.uniquemember**. This property is optional, but if it is specified, the LDAP access is faster. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
ibm.appcenter.ldap.user.loginName	The login name attribute when you use LDAP without VMM. See "Configuring LDAP ACL management (Liberty Profile)" on page 156 and "Configuring LDAP ACL management (Apache Tomcat)" on page 162.
ibm.appcenter.ldap.vmm.active	Specifies whether LDAP is done through VMM. Set to `true` to enable or `false` to disable. See "Configuring LDAP ACL management (WebSphere Application Server V8.x)" on page 152 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 149.
ibm.appcenter.ldap.vmm.adminpwd	The password when LDAP is done through VMM. See "Configuring LDAP ACL management (WebSphere Application Server V8.x)" on page 152 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 149.
ibm.appcenter.ldap.vmm.adminuser	The user when LDAP is done through VMM. See "Configuring LDAP ACL management (WebSphere Application Server V8.x)" on page 152 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 149.
ibm.appcenter.logging.formatjson	This property has only an effect when **ibm.appcenter.logging.tosystemerror** is set to `true`. If enabled, it formats JSON responses in logging messages that are directed to System.Error. Setting this property is helpful when you debug the server.

Table 38. List of the JNDI properties for the Application Center (continued)

Property	Description
ibm.appcenter.logging.tosystemerror	Specifies whether all logging messages are also directed to System.Error. Setting this property is helpful when you debug the server.
ibm.appcenter.openjpa.Log	This property is passed to OpenJPA and enables JPA logging. For details, see the Apache OpenJPA User's Guide.
ibm.appcenter.proxy.host	If the Application Center server is behind a firewall or reverse proxy, this property specifies the address of the host. Setting this property allows a user outside the firewall to reach the Application Center server. Typically, this property is the address of the proxy. See "Defining the endpoint of the application resources" on page 165.
ibm.appcenter.proxy.port	If the Application Center server is behind a firewall or reverse proxy, this property specifies the address of the host. Setting this property allows a user outside the firewall to reach the Application Center server. Typically, this property is the port of the proxy, for example 443. It is only needed if the protocol of the external and of the internal URI are different. See "Defining the endpoint of the application resources" on page 165.
ibm.appcenter.proxy.protocol	If the Application Center server is behind a firewall or reverse proxy, this property specifies the protocol (http or https). Setting this property allows a user outside the firewall to reach the Application Center server. Typically, this property is set to the protocol of the proxy. For example, **appcntr.net**. This property is only needed if the protocol of the external and of the internal URI are different. See "Defining the endpoint of the application resources" on page 165.
ibm.appcenter.proxy.scheme	This property is just an alternative name for **ibm.appcenter.proxy.protocol**.
ibm.appcenter.push.schedule.period.amount	Specifies the time schedule when you send push notifications of application updates. When applications are frequently changed on the server, set this property to send batches of notifications. For example, send all notifications that happened within the past hour, instead of sending each individual notification.
ibm.appcenter.push.schedule.period.unit	Specifies the unit for the time schedule when you send push notifications of application updates.
ibm.appcenter.services.endpoint	Enables the Application Center console to locate the Application Center REST services. Specify the external address and context root of the applicationcenter.war web application. 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. For example, https://appcntr.net:443/applicationcenter. See "Defining the endpoint of the application resources" on page 165.

Table 38. List of the JNDI properties for the Application Center (continued)

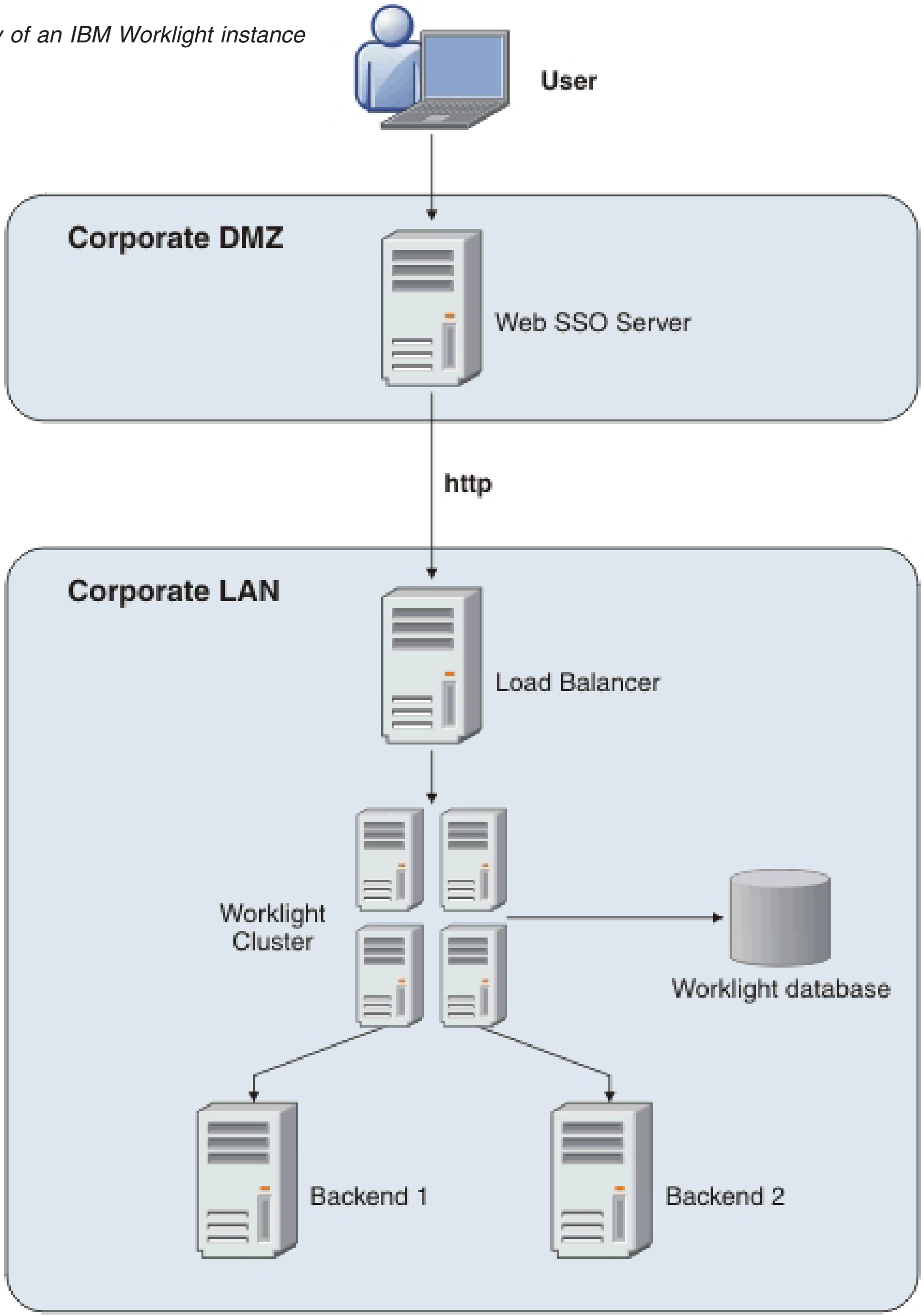
Property	Description
`ibm.appcenter.services.iconCacheMaxAge`	Specifies the amount of time in seconds cached icons remain valid for the Application Center Console and the Client. Application icons rarely change, therefore they are cached. Specify values larger than 600 (10 min) to reduce the amount of data transfer for the icons.

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 12. 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 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 13:

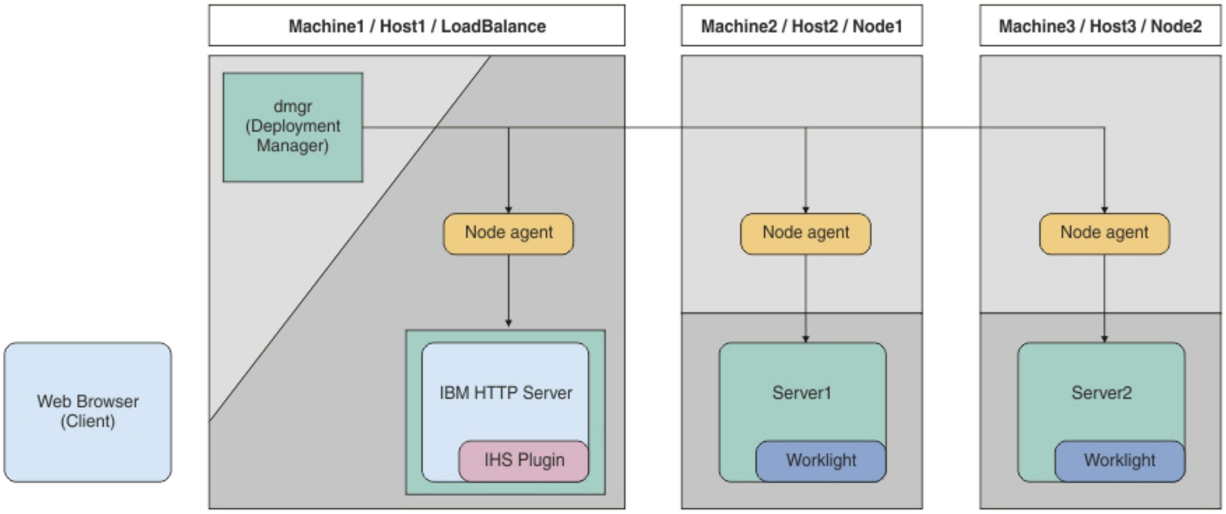


Figure 13. IBM Worklight cluster topology with IBM WebSphere Application Server Network Deployment

This procedure uses the hardware listed in Table 39 and the software listed in Table 40 on page 181.

Table 39. 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
Host4	RHEL 6.2	DB2 server

Table 40. 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.
IBM HTTP Server 8.5	IBM HTTP Server. You need to get the installation repository before start. It 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 start. It is also included in the WebSphere Application Server installation repository.
IBM Worklight V6.1.0	IBM Worklight runtime. You need to get access to the installation repository before start.
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:

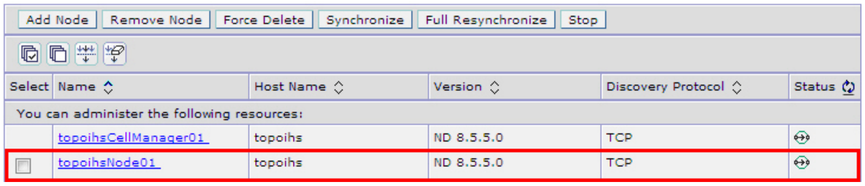
On Windows:

```
./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:					
	topoihsCellManager01	topoihs	ND 8.5.5.0	TCP	⇔
□	topoihsNode01	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:

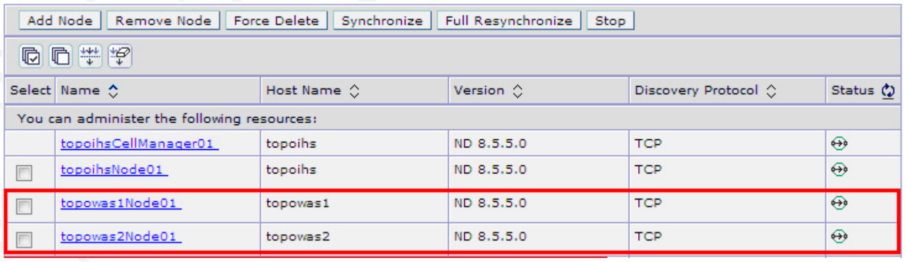
```
./addNode.bat Host1 8879 -profileName node1
```

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.

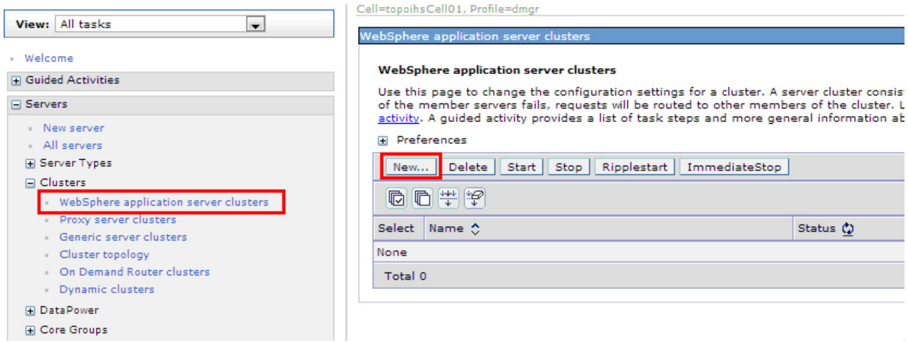


Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	topoihsCellManager01	topoihs	ND 8.5.5.0	TCP	↔
□	topoihsNode01	topoihs	ND 8.5.5.0	TCP	↔
□	topowas1Node01	topowas1	ND 8.5.5.0	TCP	↔
□	topowas2Node01	topowas2	ND 8.5.5.0	TCP	↔

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.



Cell=topoihsCell01, Profile=dmgr

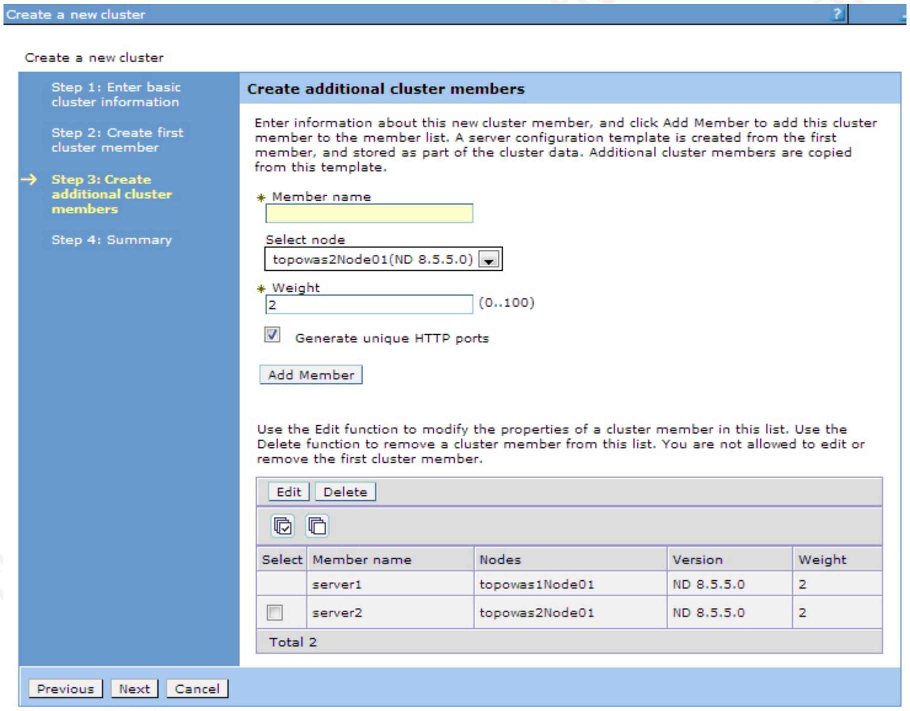
WebSphere application server clusters

WebSphere application server clusters

Use this page to change the configuration settings for a cluster. A server cluster consis
of the member servers fails, requests will be routed to other members of the cluster. L
activity. A guided activity provides a list of task steps and more general information ab

Select	Name	Status
None		
Total 0		

b. For each Worklight node, add a member to the cluster: in the fields provided, enter the required information, and then click **Add Member**.


Create a new cluster
?
Step 1: Enter basic cluster information
Step 2: Create first cluster member
Step 3: Create additional cluster members
Step 4: Summary
Add Member
Edit
Delete
Previous
Next
Cancel

Create a new cluster

Create additional cluster members

Enter information about this new cluster member, and click Add Member to add this cluster member to the member list. A server configuration template is created from the first member, and stored as part of the cluster data. Additional cluster members are copied from this template.

* Member name:

Select node: topowas2Node01(ND 8.5.5.0)

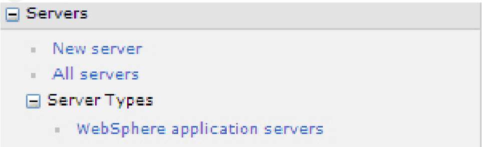
* Weight: 2 (0..100)

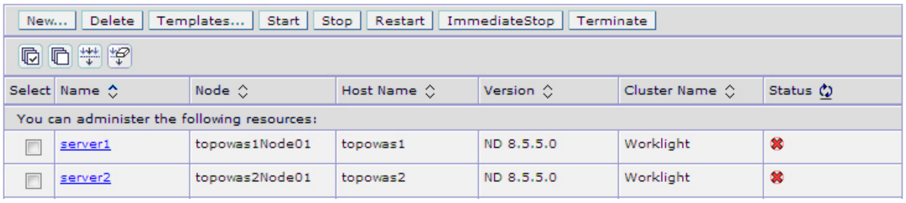
☑ Generate unique HTTP ports

Use the Edit function to modify the properties of a cluster member in this list. Use the Delete function to remove a cluster member from this list. You are not allowed to edit or remove the first cluster member.

Select	Member name	Nodes	Version	Weight
	server1	topowas1Node01	ND 8.5.5.0	2
□	server2	topowas2Node01	ND 8.5.5.0	2
Total 2				

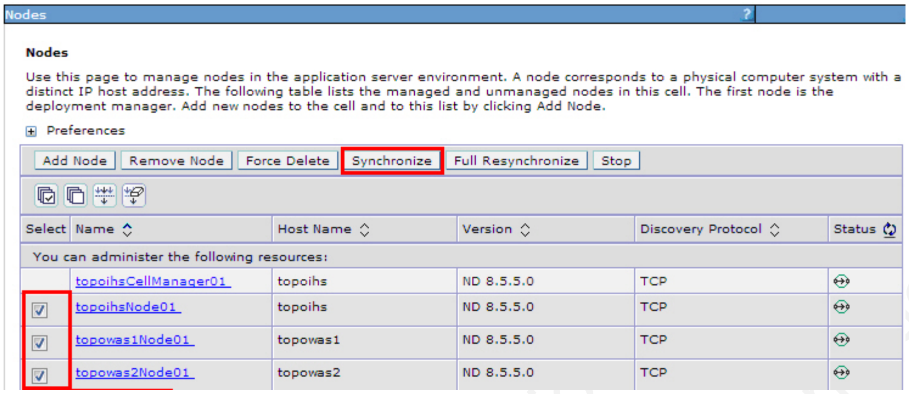
c. From the WebSphere Application Server administrative console, click **Servers** > **Server Types** > **WebSphere application servers** to check that the cluster member servers are listed.





Select	Name	Node	Host Name	Version	Cluster Name	Status
You can administer the following resources:						
□	server1	topowas1Node01	topowas1	ND 8.5.5.0	Worklight	✖
□	server2	topowas2Node01	topowas2	ND 8.5.5.0	Worklight	✖

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.



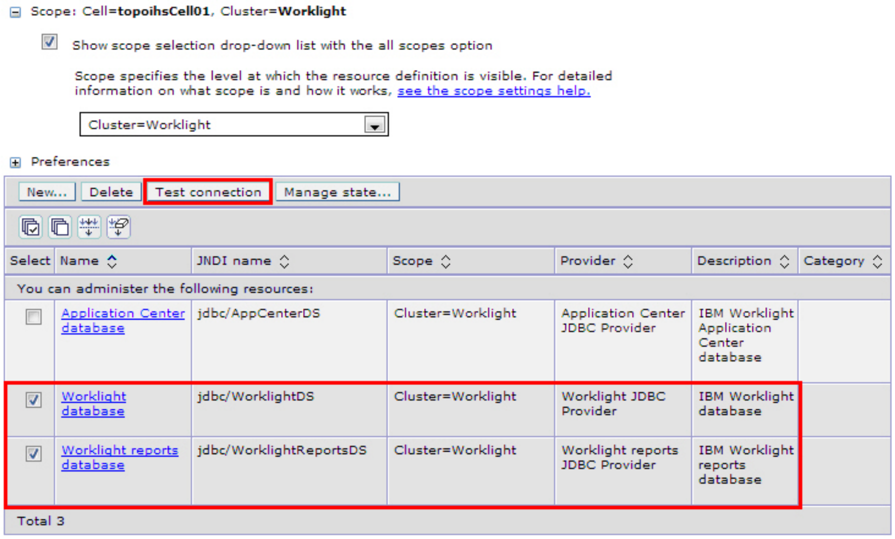
Nodes

Use this page to manage nodes in the application server environment. A node corresponds to a physical computer system with a distinct IP host address. The following table lists the managed and unmanaged nodes in this cell. The first node is the deployment manager. Add new nodes to the cell and to this list by clicking Add Node.

⊞ Preferences

Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	topoihsCellManager01	topoihs	ND 8.5.5.0	TCP	⇔
☑	topoihsNode01	topoihs	ND 8.5.5.0	TCP	⇔
☑	topowas1Node01	topowas1	ND 8.5.5.0	TCP	⇔
☑	topowas2Node01	topowas2	ND 8.5.5.0	TCP	⇔

4. Install IBM Worklight Server on Host1. Ensure that the WebSphere Application Server Network Deployment cluster is created without errors before you begin the installation. For installation instructions, see “Installing Worklight Server” on page 52.
5. Configure the databases. For instructions, see “Creating and configuring the databases with Ant tasks” on page 722.
6. In Worklight Studio, create an IBM Worklight project and build a Worklight project WAR file. See “Artifacts produced during development cycle” on page 314.
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 748. 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.



⊟ Scope: Cell=**topoihsCell01**, Cluster=**Worklight**

☑ Show scope selection drop-down list with the all scopes option

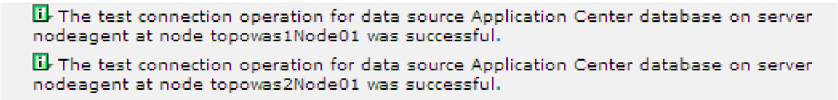
Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help.

Cluster=Worklight

⊞ Preferences

Select	Name	JNDI name	Scope	Provider	Description	Category
You can administer the following resources:						
☐	Application Center database	jdbc/AppCenterDS	Cluster=Worklight	Application Center JDBC Provider	IBM Worklight Application Center database	
☑	Worklight database	jdbc/WorklightDS	Cluster=Worklight	Worklight JDBC Provider	IBM Worklight database	
☑	Worklight reports database	jdbc/WorklightReportsDS	Cluster=Worklight	Worklight reports JDBC Provider	IBM Worklight reports database	
Total 3						

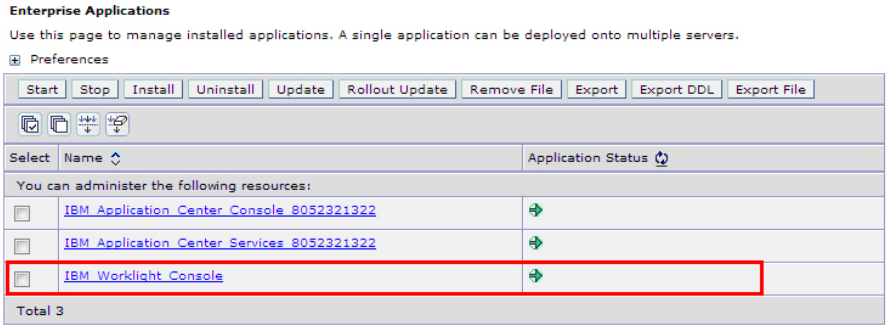
   c. Select the two data sources and click **Test connection** to verify the DB2 database connection. Confirmations similar to the ones in the following messages indicate a successful connection.



The test connection operation for data source Application Center database on server nodeagent at node topowas1Node01 was successful.

The test connection operation for data source Application Center database on server nodeagent at node topowas2Node01 was successful.

d. Go to **Applications** > **Application Types** > **WebSphere enterprise applications** and check that the Worklight Console application is running.



Enterprise Applications

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Select	Name	Application Status
You can administer the following resources:		
□	IBM Application Center Console 8052321322	➔
□	IBM Application Center Services 8052321322	➔
□	IBM Worklight Console	➔
Total 3		

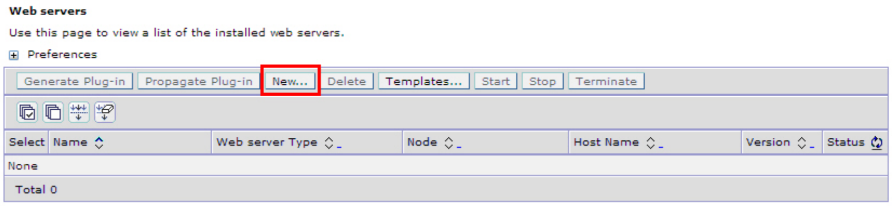
e. Now that you have deployed IBM Worklight on the two node servers, you can access the Worklight Console on each host by browsing to the associated URLs:

- `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.

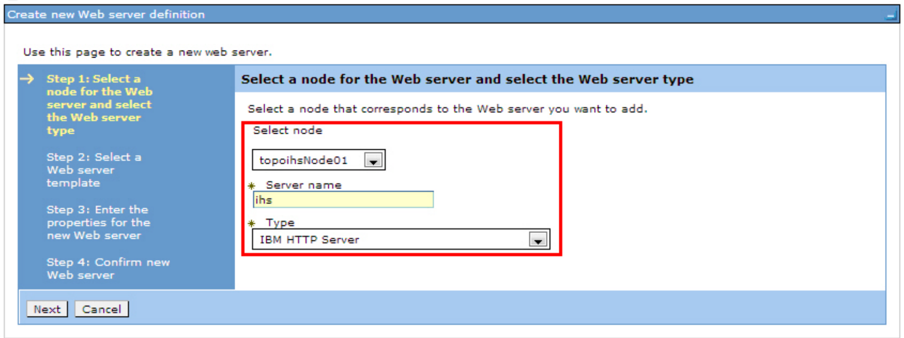


Web servers

Use this page to view a list of the installed web servers.

Select	Name	Web server Type	Node	Host Name	Version	Status
None						
Total 0						

b. Select the "ihs" node you previously created on Host1, then from the **Type** list, select **IBM HTTP Server**, and then click **Next**.



Create new Web server definition

Use this page to create a new web server.

Step 1: Select a node for the Web server and select the Web server type
Step 2: Select a Web server template
Step 3: Enter the properties for the new Web server
Step 4: Confirm new Web server
Next
Cancel

Select a node for the Web server and select the Web server type

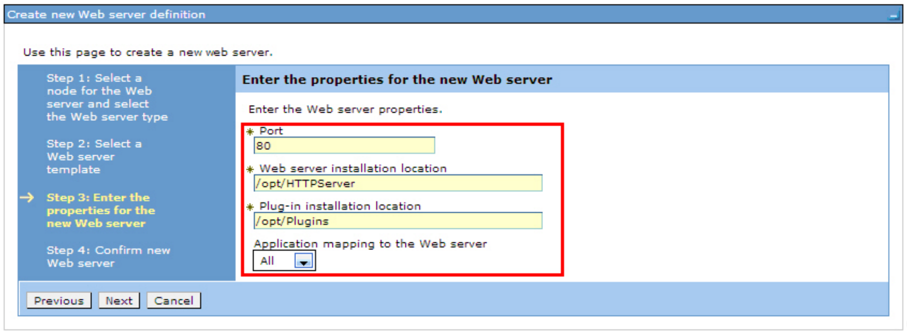
Select a node that corresponds to the Web server you want to add.

Select node: topoihsNode01

* Server name: ihs

* Type: IBM HTTP Server

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.



Create new Web server definition

Use this page to create a new web server.

Step 1: Select a node for the Web server and select the Web server type

Step 2: Select a Web server template

→ Step 3: Enter the properties for the new Web server

Step 4: Confirm new Web server

Enter the properties for the new Web server

Enter the Web server properties.

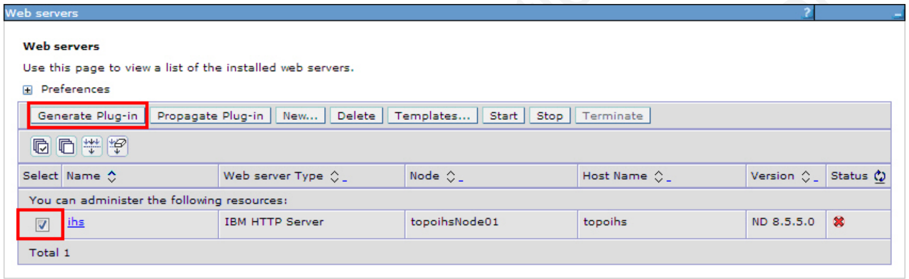
* Port: 80

* Web server installation location: /opt/HTTPServer

* Plug-in installation location: /opt/Plugins

Application mapping to the Web server: All

d. In the administrative console, on the Web servers page, click **Generate Plug-in** to generate the plug-in configuration file.



Web servers

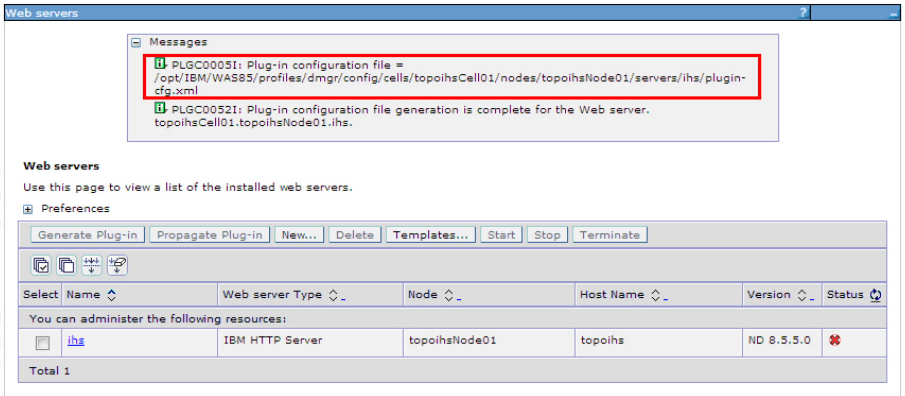
Web servers

Use this page to view a list of the installed web servers.

⊞ Preferences

Select	Name	Web server Type	Node	Host Name	Version	Status
You can administer the following resources:						
☑	ihs	IBM HTTP Server	topoihsNode01	topoihs	ND 8.5.5.0	✖
Total 1						

A confirmation message is displayed.



Web servers

⊟ Messages

PLGC0005I: Plug-in configuration file = /opt/IBM/WAS85/profiles/dmgr/config/cells/topoihsCell01/nodes/topoihsNode01/servers/ihs/plugin-cfg.xml

PLGC0052I: Plug-in configuration file generation is complete for the Web server. topoihsCell01.topoihsNode01.ihs.

Web servers

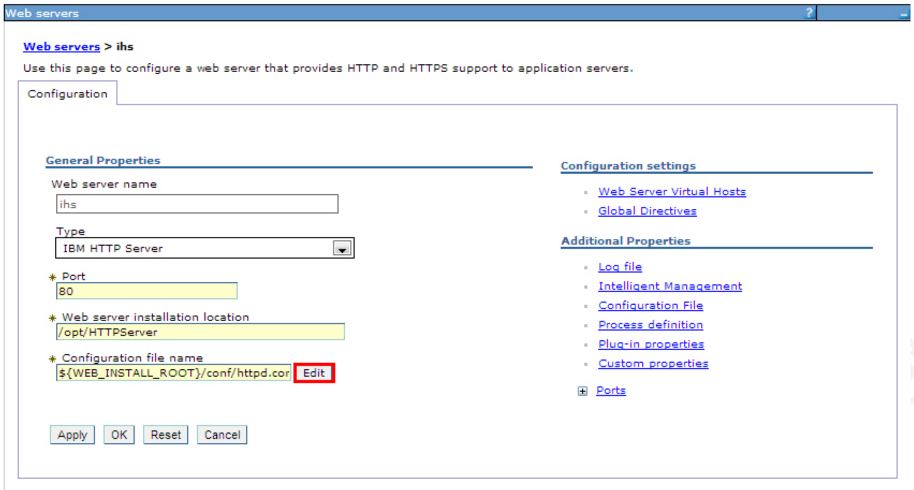
Use this page to view a list of the installed web servers.

⊞ Preferences

Select	Name	Web server Type	Node	Host Name	Version	Status
You can administer the following resources:						
□	ihs	IBM HTTP Server	topoihsNode01	topoihs	ND 8.5.5.0	✖
Total 1						

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**.



Web servers

Web servers > ihs

Use this page to configure a web server that provides HTTP and HTTPS support to application servers.

Configuration

General Properties

Web server name: ihs

Type: IBM HTTP Server

* Port: 80

* Web server installation location: /opt/HTTPServer

* Configuration file name: ${WEB_INSTALL_ROOT}/conf/httpd.cor Edit

Configuration settings

- Web Server Virtual Hosts
- Global Directives

Additional Properties

- Log file
- Intelligent Management
- Configuration File
- Process definition
- Plug-in properties
- Custom properties

Ports

Apply OK Reset Cancel

g. In the editor, add a `was_ap22_module` and a `WebSpherePluginConfig` configuration to your `http.conf` file by adding the following text:

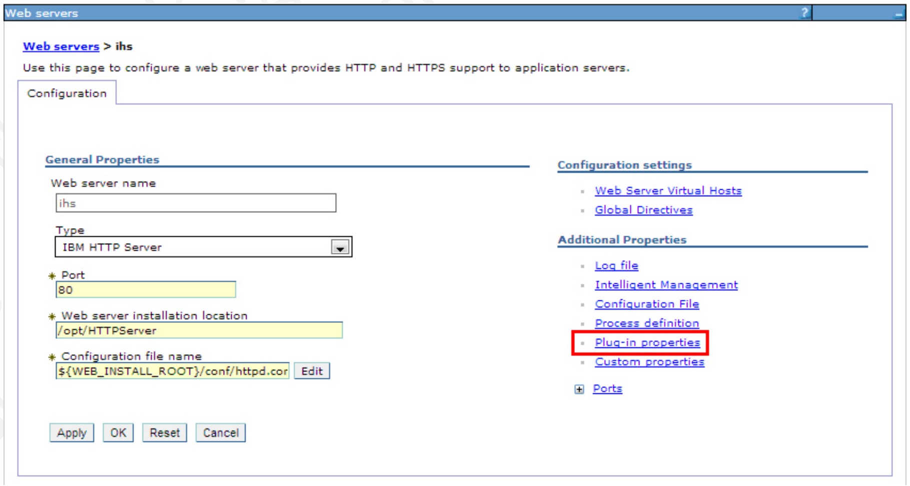
On Windows:

```
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**.



Web servers

Web servers > ihs

Use this page to configure a web server that provides HTTP and HTTPS support to application servers.

Configuration

General Properties

Web server name: ihs

Type: IBM HTTP Server

* Port: 80

* Web server installation location: /opt/HTTPServer

* Configuration file name: ${WEB_INSTALL_ROOT}/conf/httpd.cor Edit

Configuration settings

- Web Server Virtual Hosts
- Global Directives

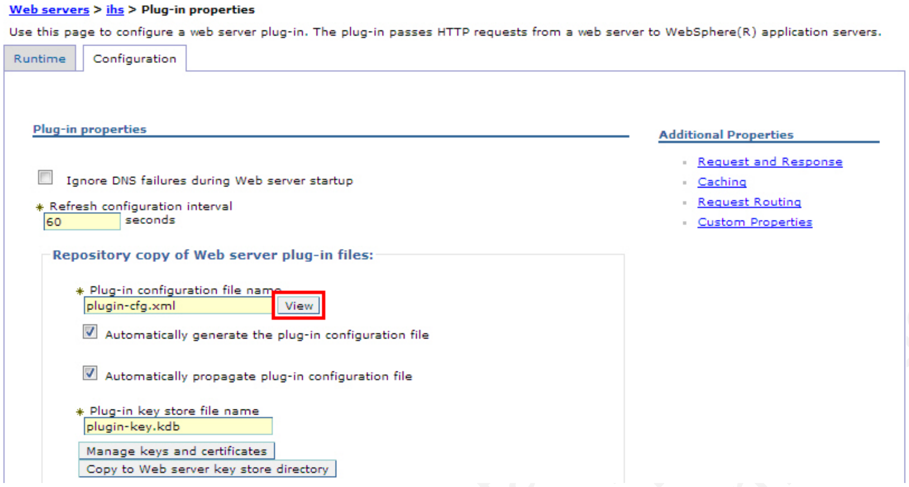
Additional Properties

- Log file
- Intelligent Management
- Configuration File
- Process definition
- Plug-in properties
- Custom properties

Ports

Apply OK Reset Cancel

i. In the **Plug-in Configuration file name** field, click **View**.



Web servers > ihs > Plug-in properties

Use this page to configure a web server plug-in. The plug-in passes HTTP requests from a web server to WebSphere(R) application servers.

Plug-in properties

□ Ignore DNS failures during Web server startup

* Refresh configuration interval
60 seconds

Repository copy of Web server plug-in files:

* Plug-in configuration file name
plugin-cfg.xml

☑ Automatically generate the plug-in configuration file

☑ Automatically propagate plug-in configuration file

* Plug-in key store file name
plugin-key.kdb

Manage keys and certificates
Copy to Web server key store directory

Additional Properties

- Request and Response
- Caching
- Request Routing
- Custom Properties

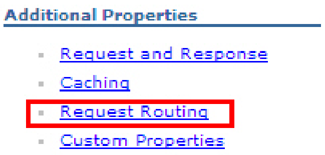
j. Search for the cluster node and Worklight URI in the `plugin-cfg.xml` file. For example:

```
<ServerCluster CloneSeparatorChange="false"
  GetDWLMTable="false"
  IgnoreAffinityRequests="true"
  LoadBalance="Round Robin"
  Name="Worklight"
  PostBufferSize="0"
  PostSizeLimit="-1"
  RemoveSpecialHeaders="true"
  RetryInterval="60"
  ServerIOTimeoutRetry="-1">
  <Server CloneID="17oi9lu2o"
    ConnectTimeout="5"
    ExtendedHandshake="false"
    LoadBalanceWeight="2"
    MaxConnections="-1"
    Name="topowas1Node01_server1"
    ServerIOTimeout="900"
    WaitForContinue="false">
    <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"
    LoadBalanceWeight="2"
    MaxConnections="-1"
    Name="topowas2Node01_server2"
    ServerIOTimeout="900"
    WaitForContinue="false">
    <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="/appcenterconsole/*"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/worklight/*"/>
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/applicationcen
</UriGroup>
```

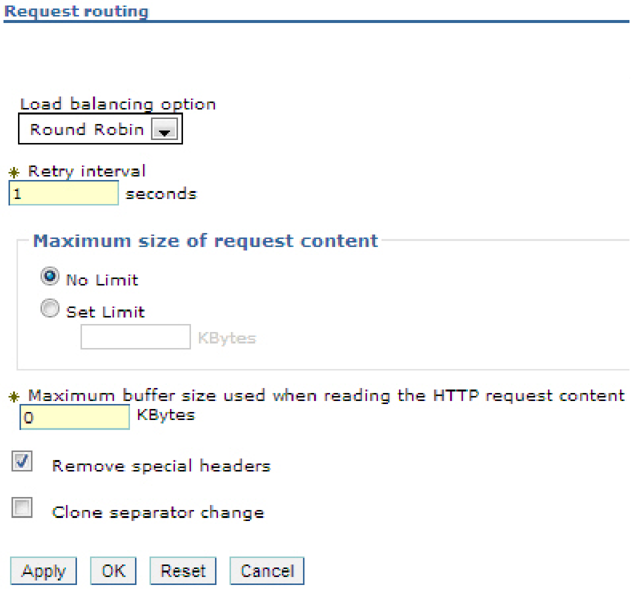
If your configuration file does not include cluster servers and URIs, delete the "ihs" server and create it again.

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

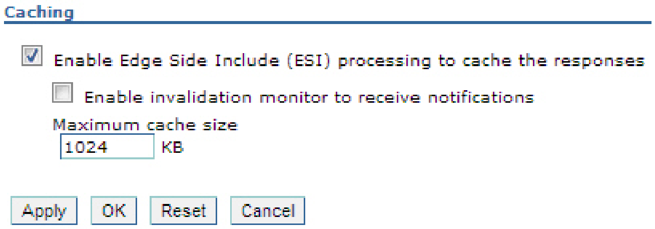
○ Set Limit
KBytes

* Maximum buffer size used when reading the HTTP request content
0 KBytes

☑ Remove special headers

☐ Clone separator change

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

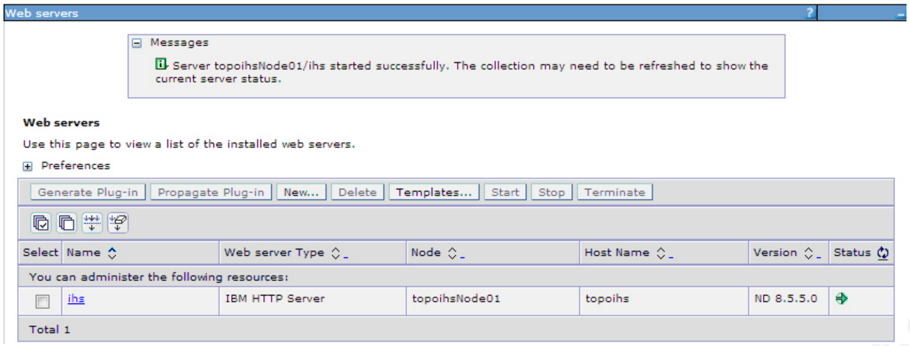
☐ Enable invalidation monitor to receive notifications

Maximum cache size
1024 KB

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**.



Server topoihsNode01/ihs started successfully. The collection may need to be refreshed to show the current server status.

Web servers

Use this page to view a list of the installed web servers.

Preferences

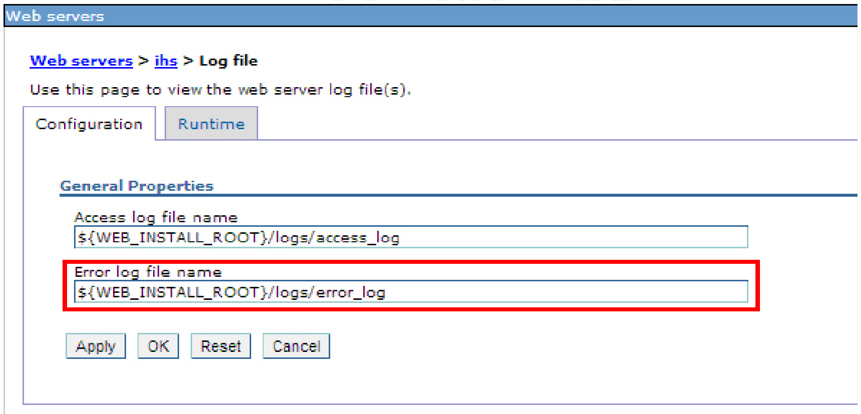
Select	Name	Web server Type	Node	Host Name	Version	Status
You can administer the following resources:						
□	ihs	IBM HTTP Server	topoihsNode01	topoihs	ND 8.5.5.0	➔
Total 1						

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.



Web servers > ihs > Log file

Use this page to view the web server log file(s).

General Properties

Access log file name
${WEB_INSTALL_ROOT}/logs/access_log

Error log file name
${WEB_INSTALL_ROOT}/logs/error_log

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 14 on page 192:

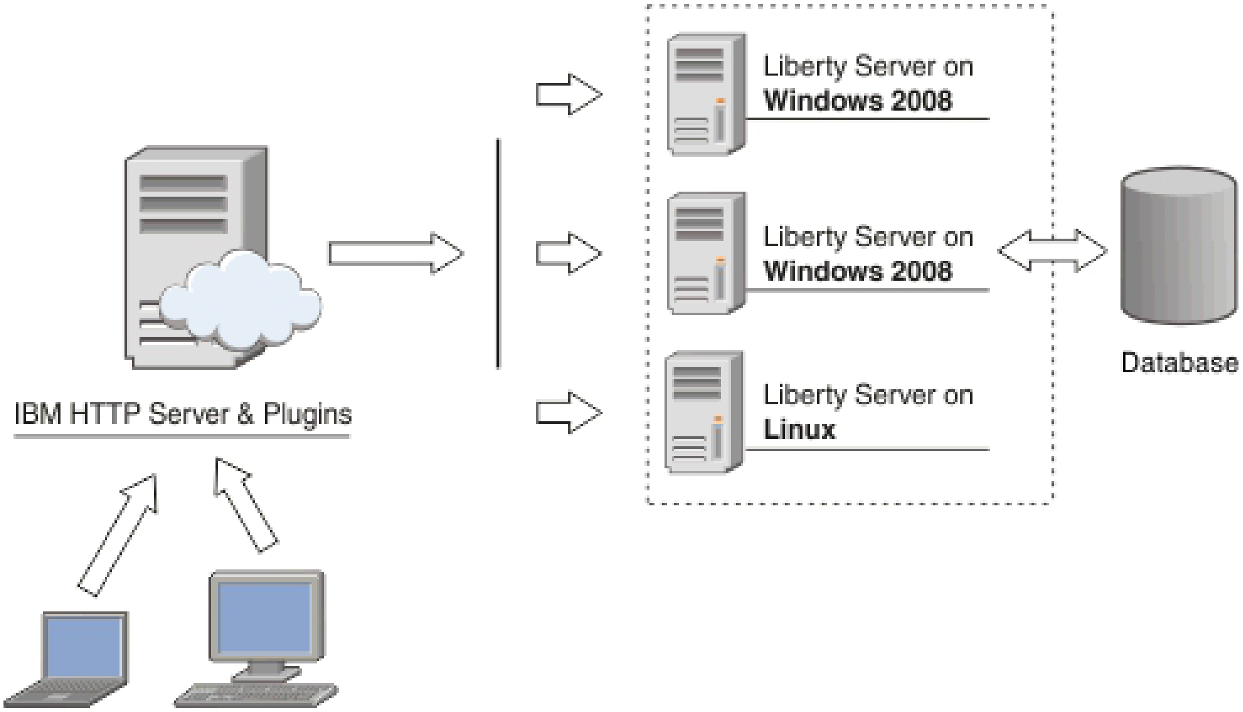


Figure 14. IBM Worklight cluster topology with Liberty Profile

This procedure uses the hardware listed in Table 41 and the software listed in Table 42.

Table 41. 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 42. 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 start. It 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 start. It is also included in the WebSphere Application Server installation repository.

Table 42. Software (continued)

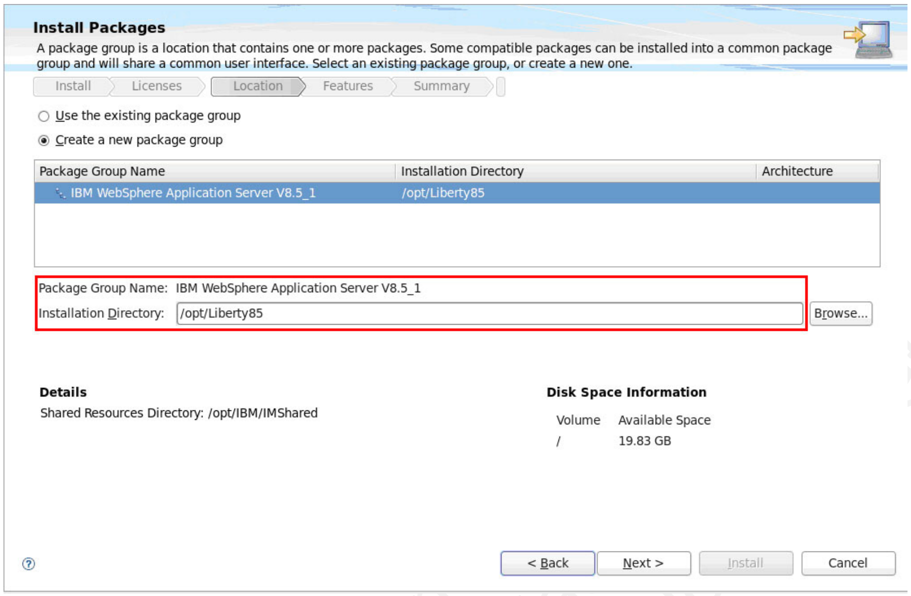
Name	Description
IBM Liberty Profile 8.5	Liberty server. You need to get the installation repository before you start. It is also included in the WebSphere Application Server installation repository.
IBM Worklight V6.1.0	IBM Worklight runtime. You need to get access to the installation repository before start.
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**.



Install Packages

A package group is a location that contains one or more packages. Some compatible packages can be installed into a common package group and will share a common user interface. Select an existing package group, or create a new one.

○ Use the existing package group

● Create a new package group

Package Group Name	Installation Directory	Architecture
IBM WebSphere Application Server V8.5_1	/opt/Liberty85	

Package Group Name: IBM WebSphere Application Server V8.5_1

Installation Directory: /opt/Liberty85

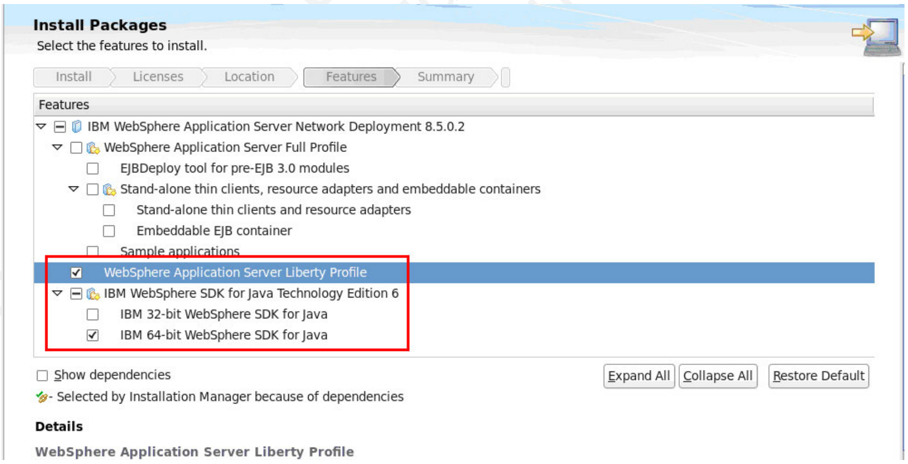
Details

Shared Resources Directory: /opt/IBM/IMShared

Disk Space Information

Volume	Available Space
/	19.83 GB

d. Select **WebSphere Application Server Liberty Profile** clear **WebSphere Application Server Full Profile**, and then click **Next** repeatedly until the installation is complete.



Install Packages

Select the features to install.

Features

- ⊟ IBM WebSphere Application Server Network Deployment 8.5.0.2
  - □ WebSphere Application Server Full Profile
    - □ EJBDeploy tool for pre-EJB 3.0 modules
    - □ Stand-alone thin clients, resource adapters and embeddable containers
      - □ Stand-alone thin clients and resource adapters
      - □ Embeddable EJB container
    - □ Sample applications
  - ☑ WebSphere Application Server Liberty Profile
  - ⊟ IBM WebSphere SDK for Java Technology Edition 6
    - □ IBM 32-bit WebSphere SDK for Java
    - ☑ IBM 64-bit WebSphere SDK for Java

□ Show dependencies

- Selected by Installation Manager because of dependencies

Details

WebSphere Application Server Liberty Profile

e. Repeat the previous steps to install another Liberty Profile server on Host3.

3. Install IBM Worklight Server on Host2 and Host3. For instructions, see “Installing Worklight Server” on page 52.

4. Configure the IBM Worklight databases. For instructions, see “Creating and configuring the databases with Ant tasks” on page 722.

5. In Worklight Studio, create an IBM Worklight project and build a Worklight project WAR file. See “Artifacts produced during development cycle” on page 314.

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 748. 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 199.

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>
    ```

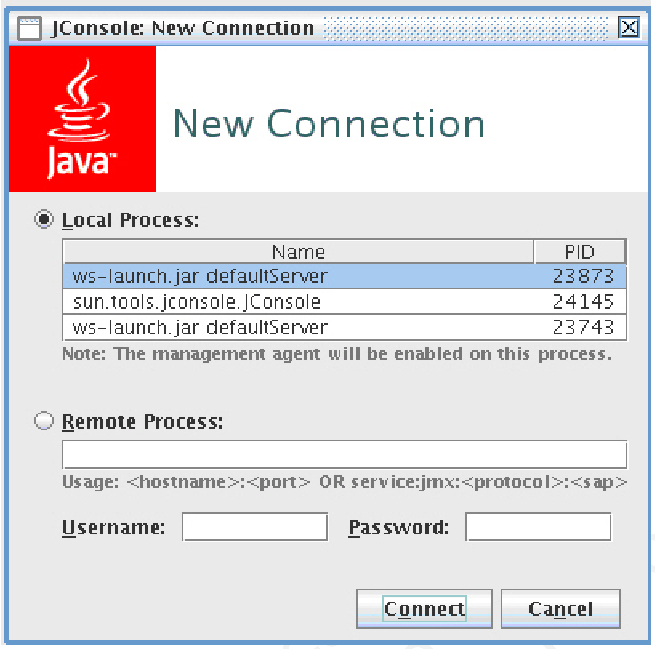
    b. Use one of the following methods to generate the `plugin-cfg.xml` file for the Liberty server running your application.

    - 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**.



● Local Process:

Name	PID
ws-launch.jar defaultServer	23873
sun.tools.jconsole.JConsole	24145
ws-launch.jar defaultServer	23743

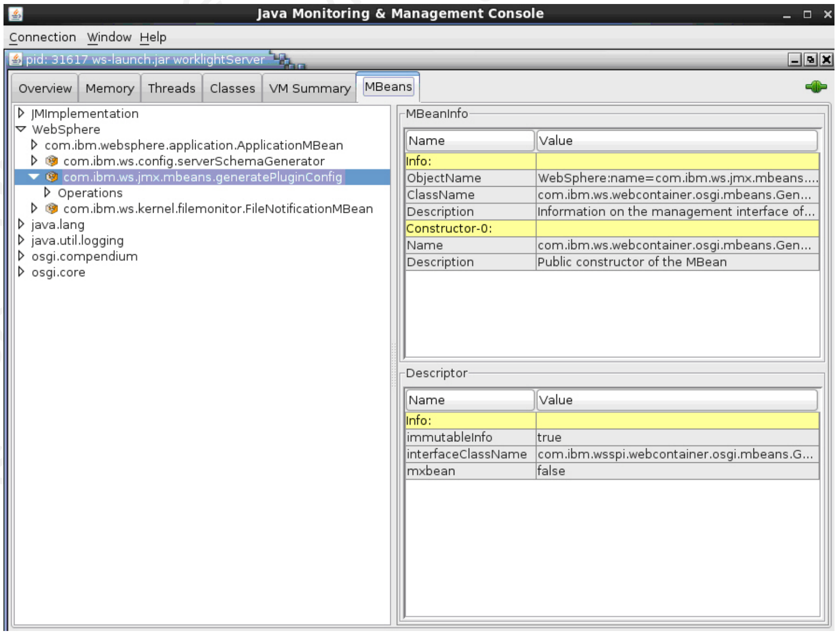
Note: The management agent will be enabled on this process.

○ Remote Process:

Usage: <hostname>:<port> OR service:jmx:<protocol>:<sap>

Username: Password:

3) In the Java Monitoring & Management Console, click the **MBeans** tab.


Java Monitoring & Management Console
Connection Window Help
pid: 31617 ws-launch.jar worklightServer
Overview
Memory
Threads
Classes
VM Summary
MBeans

- JMImplementation
- WebSphere
  - com.ibm.websphere.application.ApplicationMBean
  - com.ibm.ws.config.serverSchemaGenerator
  - com.ibm.ws.jmx.mbeans.generatePluginConfig
    - Operations
  - com.ibm.ws.kernel.filemonitor.FileNotificationMBean
- java.lang
- java.util.logging
- osgi.compendium
- osgi.core

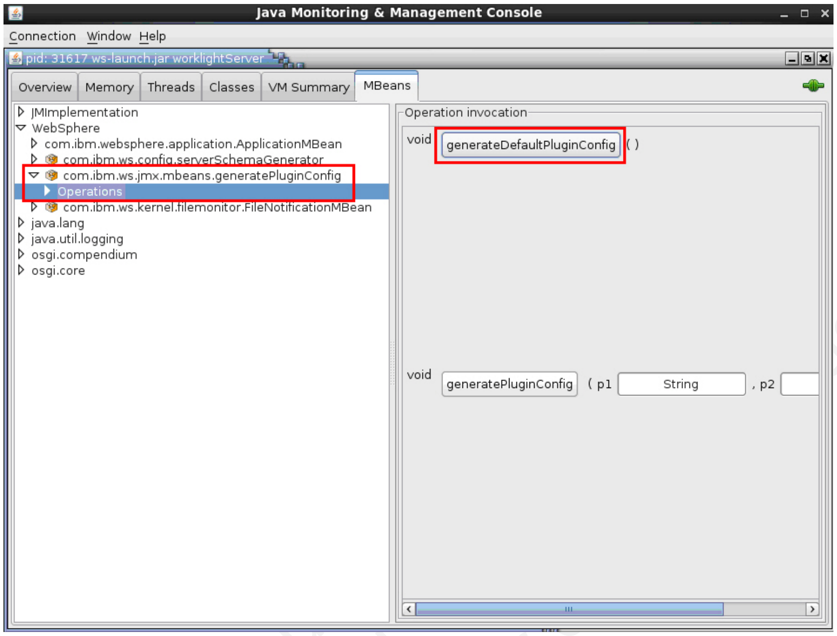
MBeanInfo

Name	Value
Info:	
ObjectName	WebSphere:name=com.ibm.ws.jmx.mbeans....
ClassName	com.ibm.ws.webcontainer.osgi.mbeans.Gen...
Description	Information on the management interface of...
Constructor-0:	
Name	com.ibm.ws.webcontainer.osgi.mbeans.Gen...
Description	Public constructor of the MBean

Descriptor

Name	Value
Info:	
immutableInfo	true
interfaceClassName	com.ibm.wsspi.webcontainer.osgi.mbeans.G...
mxbean	false

4) Select and invoke the defaultPluginConfig generation MBean operation to generate the `plugin-cfg.xml` file.


Java Monitoring & Management Console
Connection Window Help
pid: 31617 ws-launch.jar worklightServer
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
WebSphere
com.ibm.websphere.application.ApplicationMBean
com.ibm.ws.config.serverSchemaGenerator
com.ibm.ws.jmx.mbeans.generatePluginConfig
Operations
com.ibm.ws.kernel.filemonitor.FileNotificationMBean
java.lang
java.util.logging
osgi.compendium
osgi.core
Operation invocation
void
generateDefaultPluginConfig
( )
void
generatePluginConfig
(p1
String
, p2

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="HostHeader"
  ChunkedResponse="false"
  FIPSEnable="false"
  IISDisableNagle="false"
  IISPluginPriority="High"
  IgnoreDNSFailures="false"
  RefreshInterval="60"
  ResponseChunkSize="64"
  SSLConsolidate="false"
  SSLPKCSDriver="REPLACE"
  SSLPKCSPassword="REPLACE"
  TrustedProxyEnable="false"
  VHostMatchingCompat="false">
  <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"
    IgnoreAffinityRequests="true"
    LoadBalance="Round Robin"
    Name="String_default_node_Cluster"
    PostBufferSize="64"
    PostSizeLimit="-1"
    RemoveSpecialHeaders="true"
    RetryInterval="60">
    <Server CloneID="s56"
      ConnectTimeout="0"
      ExtendedHandshake="false"
```

```
   MaxConnections="-1"
   Name="default_node_String0"
   ServerIOTimeout="900"
   WaitForContinue="false">
    <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-web
 </UriGroup>
 <Route ServerCluster="String_default_node_Cluster"
  UriGroup="default_host_String_default_node_Cluster_URIs"
  VirtualHostGroup="default_host"/>
</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.
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"
  LoadBalance="Round Robin"
  Name="String_default_node_Cluster1"
  PostBufferSize="64"
  PostSizeLimit="-1"
  RemoveSpecialHeaders="true"
```

```
    RetryInterval="60">
    <Server CloneID="s59"
      ConnectTimeout="0"
      ExtendedHandshake="false"
      MaxConnections="-1"
      Name="default_node_String1"
      ServerIOTimeout="900"
      WaitForContinue="false">
      <Transport Hostname="wasvm59.example.com" Port="9080" Protocol="http"/>
    </Server>
    <Server CloneID="s56"
      ConnectTimeout="0"
      ExtendedHandshake="false"
      MaxConnections="-1"
      Name="default_node_String2"
      ServerIOTimeout="900"
      WaitForContinue="false">
      <Transport Hostname="wasvm56.example.com" Port="9080" Protocol="http"/>
    </Server>
    <Server CloneID="vm28"
      ConnectTimeout="0"
      ExtendedHandshake="false"
      MaxConnections="-1"
      Name="default_node_String3"
      ServerIOTimeout="900"
      WaitForContinue="false">
      <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:

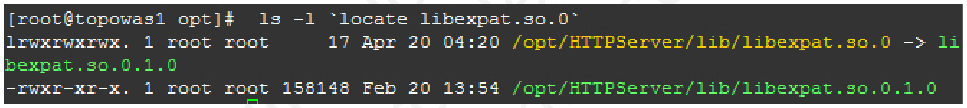
   ```
   ldd /opt/HTTPServer/bin/httpd
   ```

   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 has been previously shown.

      ```
      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)
libfreebl3.so => /lib64/libfreebl3.so (0x0000003a08000000)
```

6. Start the IBM HTTP Server.

Integrating IBM WebSphere DataPower with a cluster of Worklight Servers

You can use IBM WebSphere DataPower as a gateway for all incoming connections for IBM Worklight and Application Center, and IBM HTTP Server (IHS) for load-balancing IBM Worklight Servers that are deployed on an IBM WebSphere Application Server 8.5 cluster or a Liberty Profile server farm.

Before you begin

Ensure that the following environments are available:

- Worklight Server is deployed on an IBM WebSphere Application Server ND cluster or on a Liberty Profile server farm and is configured to use DB2 or any compatible database. For more information, see "Typical topologies of an IBM Worklight instance" on page 178.
- Application Center is set up on an IBM WebSphere Application Server ND cluster. For more information, see "Installing the Application Center" on page 138.
- IBM WebSphere DataPower XI52.
- IBM HTTP Server.
- Any LDAP server with SSL enabled.

About this task

This procedure explains how to set up IBM Worklight in the topology similar to the one shown in Figure 15 on page 202.

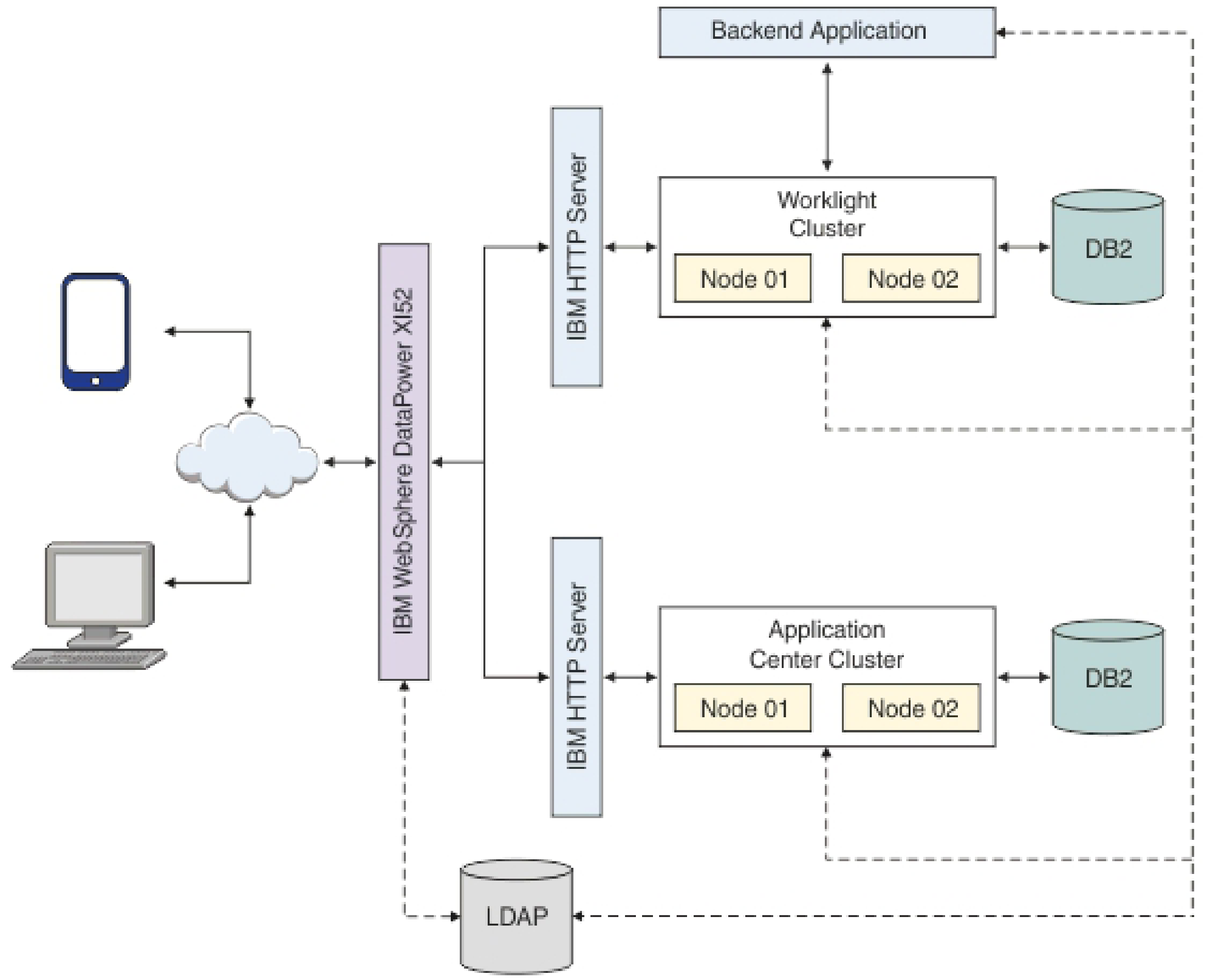


Figure 15. IBM Worklight integration with an IBM WebSphere Application Server 8.5 Cluster or a Liberty Profile Server Farm

.

DataPower XI52 acts as the gateway for all IBM Worklight and Application Center requests. DataPower validates all incoming user credentials against an LDAP registry. If the validation is successful, DataPower generates an LTPA token, which is present as part of a session cookie. This cookie is only valid for one session and is used for all further requests during that session. The cookies themselves contain information about the user that has been authenticated, the realm for which the user was authenticated (such as an LDAP server) and a timestamp. A request with a valid LTPA cookie can access a server that is a member of the same authentication domain as the first server. The request is automatically authenticated, thereby enabling single-sign-on (SSO).

All requests that reach the Worklight cluster or the backend application validate only the LTPA token. If the LTPA token is valid, the request is authenticated according to the rules that are set. The LTPA token guarantees that as long as the token is valid, all requests have SSO capability into all backend servers, including IBM Worklight and Application center.

The following sequence of events takes place when a mobile application makes a request (see Figure 16):

1. The mobile application makes a request to the DataPower gateway.
2. DataPower checks for an LTPA token in the incoming request.
3. If a valid LTPA token is present, the request is sent to the IBM Worklight cluster.
   - If an LTPA token is not present or if the token is not valid, DataPower throws an authentication challenge. The mobile application handles the challenge and then prompts for user credentials.
4. The Worklight cluster validates the LTPA token and sends the request to the backend application server along with the LTPA token.
5. The backend application server validates the LTPA token and sends the response back to IBM Worklight.
6. IBM Worklight forwards the request to DataPower, and DataPower forwards it to the requesting mobile application.

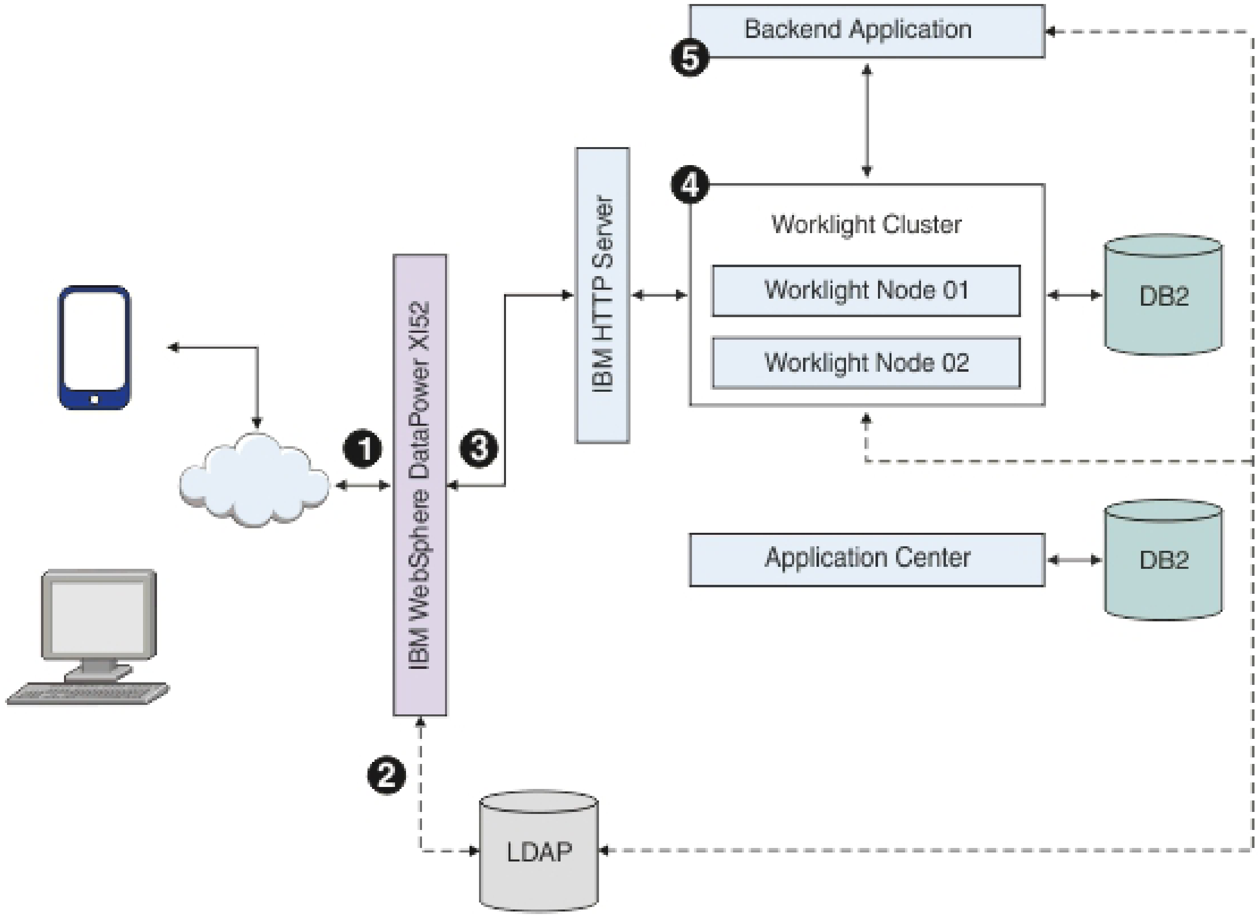


Figure 16. Mobile application request-response flow

The Application Center request-response flow takes a similar route to the mobile application flow, except that requests are routed to the Application Center server instead of to the Worklight cluster (see Figure 17 on page 204).

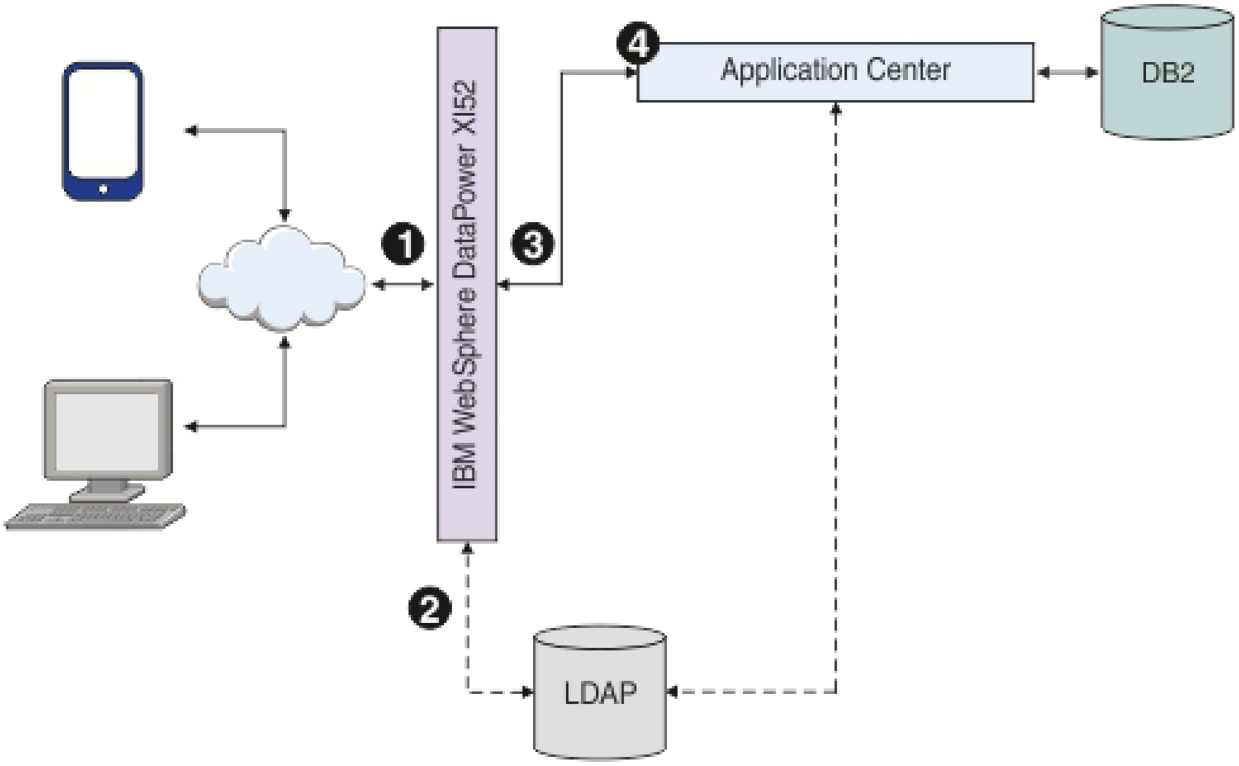


Figure 17. Application Center request-response flow

Procedure

1. Configure server security.
   - On a WebSphere Application Server cluster:
     a. Login to the WebSphere Application Server integrated solutions console.
     b. Enable and configure application security.
        1) Navigate to **Security** > **Global security**, and then click **Security Configuration Wizard**.
        2) In the "Specify extent of protection" pane, select **Enable application security**.
        3) In the "Select user repository" pane, click **Federated repositories** to integrate with the LDAP server. Several different repositories, both LDAP and non-LDAP, can be configured in the federated repository. Enter the LDAP server details. Refer to the WebSphere Application Server documentation for detailed instructions.
        4) Complete the configuration steps and save your changes.
        5) On the "Global security" page, confirm that the following settings apply:
           - The **Enable administrative security** is selected.
           - The user account repository is set to LDAP.

     c. Enable WebSphere Application Server LTPA SSO between the Worklight Server cluster and backend servers. To support SSO across multiple WebSphere Application Server domains or cells, you must share the LTPA keys and the password among the domains. You need to export the LTPA keys from one of the domains and import them into all other domains in which you want to enable SSO. For detailed instructions, see Configuring LTPA and working with keys.

   d. Stop and restart the WebSphere Application Server cluster for the application security changes to take effect.

- On a Liberty Profile server farm:

   a. Integrate the LDAP server with Liberty Profile, For detailed instructions, see Configuring LDAP user registries with the Liberty profile. You must configure LDAP user registries on each member of the liberty server farm. The following file is a sample LDAP configuration for Liberty server:

```
<!-- LDAP configuration Start -->
<ldapRegistry id="IBMDirectoryServerLDAP" realm="WASLTPARealm"
              host="9.186.9.169" port="389" ignoreCase="true"
              baseDN="dc=worklight,dc=com"
              bindDN="cn=admin,dc=worklight,dc=com"
              bindPassword="passw0rd"
              ldapType="IBM Tivoli Directory Server">
<idsFilters userFilter="(&amp;(uid=%v)(objectclass=posixAccount))"
            groupFilter="(&amp;(cn=%v)(objectclass=posixGroup))"
            userIdMap="*:uid"
            groupIdMap="*:cn"
            groupMemberIdMap="ibm-allGroups:member;ibm-allGroups:uniqueMember;groupOfName
</ldapRegistry>
```

   b. Configure SSO for the Liberty server farm. To enable SSO on Liberty, you must configure an LTPA key file for each Liberty server in the Liberty farm. See Configuring LTPA on the Liberty profile. The following file is a sample LTPA configuration for Liberty server:

```
<ltpa keysFileName="${server.config.dir}/resources/security/ltpa.keystore" keysPassword=
```

2. Configure Worklight Server.

   You can secure IBM Worklight in a typical WebSphere Application Server runtime environment in 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.

   Option 1 provides greater authentication security. The application server, such as the IBM WebSphere Application Server Liberty Profile (Liberty) protects all resources and forces users to log in before any other authentication mechanism. The behavior occurs regardless of the expected authentication order for a security test. See “Supported configurations for LTPA” on page 807 for more information.

   Once the user has been successfully authenticated, an LTPA token is returned. This LTPA token needs to be present as part of all future requests from the mobile application, including adapter invocations. On the Worklight Server side, the call to the backend application should be modified to carry this LTPA token.

   For the purpose of explaining how this is done, assume that authentication configuration has a security test that uses a realm called WASLTPARealm which is of type WebSphere LTPA, and that there is an HTTP adapter defined on the server called SecureAdapter with a procedure getAccountInfo.

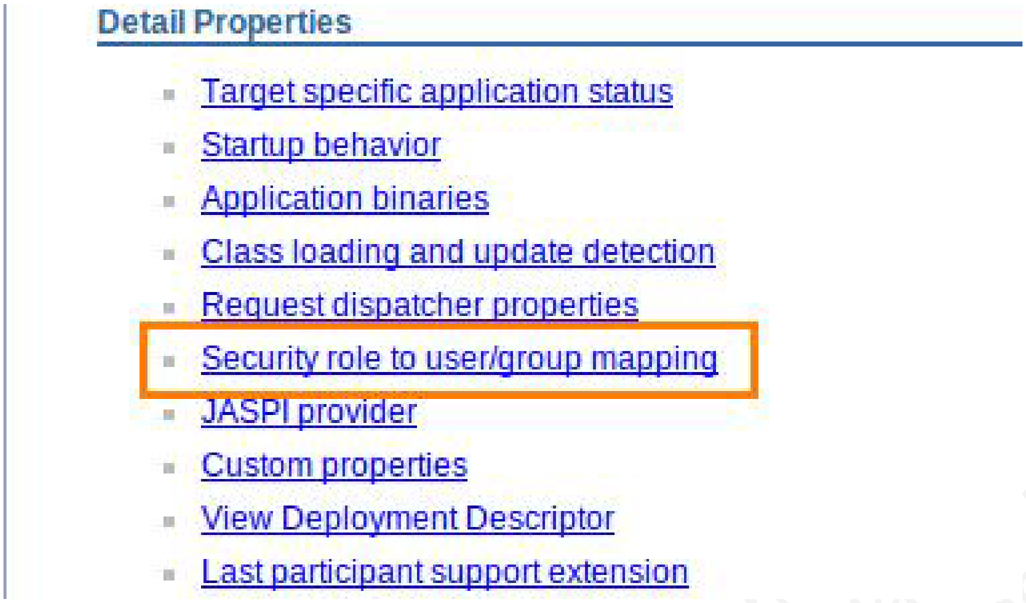
The following code snippet shows how to pass the LTPA token when invoking an adapter procedure from the mobile application.

```
function getAccountInfo(){
  var ltpaToken
  if(WL.Client.isUserAuthenticated('WASLTPARealm')){
    var attrs = WL.Client.getUserInfo('WASLTPARealm', 'attributes');
    if(attrs){
      ltpaToken = attrs.LtpaToken;
      console.log('Set ltpaToken again: '+ltpaToken);
    }
  }

  var token = {'LtpaToken2' : ltpaToken};
  var invocationData = {
    adapter: "SecureAdapter",
    procedure: "getAccountInfo",
    parameters: [token]
  };

  WL.Client.invokeProcedure(invocationData, {
    onSuccess: <on success callback>,
    onFailure: <on failure callback>
  });
}
```

On the server side, the adapter procedure needs to get the token, which is passed as a parameter. This parameter holds the LTPA token information that is used by the adapter to contact the backend service.

```
function getAccountInfo(token) {
  WL.Logger.info(token);

  var input = {
    method : 'get',
    returnedContentType : 'xml',
    cookies: token,
    path : '<path to the backend service>'
  };

  return WL.Server.invokeHttp(input);
}
```

Once this IBM Worklight WAR file has been deployed on the cluster, you need to map the users against the roles. In the WebSphere Application Server console, open the application configuration tab of the deployed Worklight Server and click **Security role to user/group mapping** to map the LDAP user to the IBM Worklight roles.



Detail Properties

- Target specific application status
- Startup behavior
- Application binaries
- Class loading and update detection
- Request dispatcher properties
- Security role to user/group mapping
- JASPI provider
- Custom properties
- View Deployment Descriptor
- Last participant support extension

Figure 18. Mapping the LDAP user to WebSphere Application Server

Select your role name and click **Map users** to map the LDAP user to this application.

For IBM Worklight V6.0 and earlier, you must edit the `web.xml` file and add the user roles. For IBM Worklight V6.1.0 and later, the roles can be added as part of the `WASLTPAModule` login module. See "WASLTPAModule login module" on page 618.

3. Configure Application Center.
   a. Complete the following configuration tasks depending on the server being used:
      - "Configuring WebSphere Application Server full profile" on page 139
      - "Configuring WebSphere Application Server Liberty Profile" on page 140
   b. Manage users with LDAP.

      Application Center uses two security roles: `appcenteradmin` and `appcenteruser`. The LDAP users need to be mapped against the security roles.

      Depending on the server that you are using, refer to the "Configuring LDAP authentication" section under one of the following documentation links:
      - "LDAP with WebSphere Application Server V7" on page 145
      - "LDAP with WebSphere Application Server V8.x" on page 150
      - "LDAP with Liberty Profile" on page 155
   c. Define the endpoint of the application resources.

      In this configuration, Application Center is behind DataPower, which is acting as a secure reverse proxy. To manage the applications on your device, the Application Center console must be able to locate the Application Center REST services and 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 Application Center JNDI properties must reference the DataPower gateway's details:

- **`ibm.appcenter.services.endpoint`**
- **`ibm.appcenter.proxy.protocol`**
- **`ibm.appcenter.proxy.host`**
- **`ibm.appcenter.proxy.port`**

Depending on the server type, set the Application Center JNDI properties by completing one of the following procedures:

- “Configuring the endpoint of the application resources (Full Profile)” on page 166
- “Configuring the endpoint of the application resources (Liberty profile)” on page 167

4. Configure DataPower. DataPower XI52 acts as the gateway for all IBM Worklight and Application Center requests. DataPower validates all incoming user credentials against an LDAP registry. The following sections show how to configure DataPower.

   a. Create a new multi-protocol gateway. Complete the following steps:

      1) From the DataPower XI52 control panel, click the **Multi-Protocol Gateway** icon to open the Multi-Protocol Gateway main page.

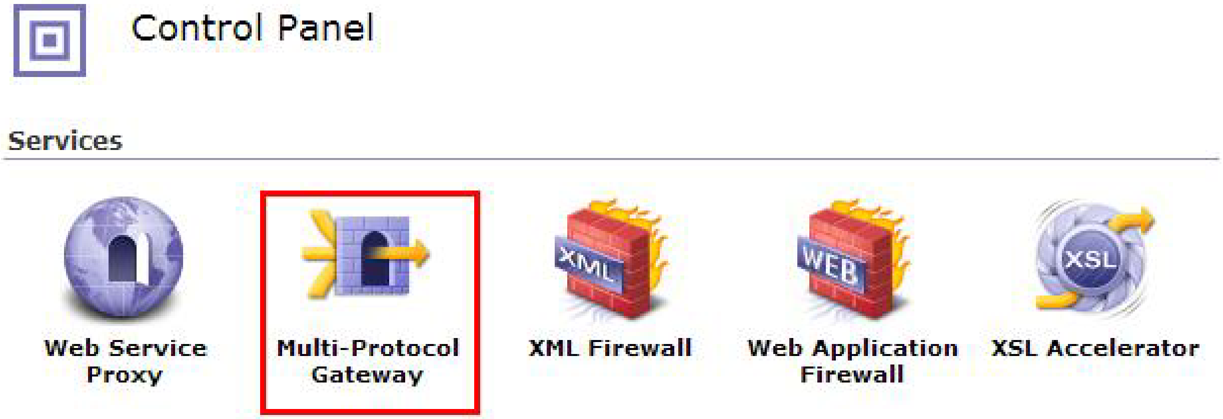


Figure 19. Accessing the Multi-Protocol Gateway main page

      2) Click **Add** to add a new gateway.

      3) Provide a name for the gateway and set **Type** to `dynamic-backend`.

      4) Make sure that **Request Type** and **Response Type** are set to `Non-XML`.

      5) On the Advanced tab page, select **Follow Redirects** and **Process Backend Errors**.

      6) On the Stylesheet Params tab page, add the parameters listed in Table 43 on page 209:

Table 43. Stylesheet parameters

Parameter name	Value
`{http://www.datapower.com/param/config}applicationcenterBackend`	`http://<appcenterHostName>:<port>`
`{http://www.datapower.com/param/config}worklightBackend`	`http://<worklightIHSHostName>:<port>`

7) On the General tab page, add an HTTPS (SSL) Front Side Handler with reverse SSL Proxy Profile. Ensure that the following methods and versions are selected:
   - HTTP 1.0
   - HTTP 1.1
   - POST method
   - GET method
   - PUT method
   - HEAD method
   - OPTIONS
   - DELETE method
   - URL with Query Strings
   - URL with Fragment Identifiers
8) Click the plus sign (+) to add a new multi-protocol gateway policy.
9) Provide a name for the policy, click **Apply Policy**, and then click **Close Window**. The policy is added to the gateway.
10) Apply your configuration.

b. Edit the multi-protocol gateway policy. Add the following rules to provide form-based authentication, generate an LTPA token and verify the LTPA token. All the rules are described in the following tables. You must list them in the same order.
   1) `worklight-ssl-policy_skipFavicon`: see Table 44
   2) `worklight-ssl-policy_verifyLTPA`: see Table 45
   3) `worklight-ssl-policy_allowSSLLoginPage`: see Table 46 on page 210
   4) `worklight-ssl-policy_worklightSSLLogin`: see Table 47 on page 211

Table 44. Properties of `worklight-ssl-policy_skipFavicon`

Property	Value
Direction	Client to Server.
Match	• Type = URL • Pattern = `/favicon.ico`
Advanced	"Set Variable" -> `var://service/mpgw/skip-backside = 1`
Result	Not applicable.

Table 45. Properties of `worklight-ssl-policy_verifyLTPA`

Property	Value
Direction	Client to Server.

Table 45. Properties of `worklight-ssl-policy_verifyLTPA` (continued)

Property	Value
Match	• Type = HTTP • HTTP tag = Cookie • Pattern = `*LtpaToken*`
AAA	• Input: INPUT • Output: NULL Add a new AAA Policy named `VerifyLTPA` with the following configuration: • Extract Identity: LTPA token • Method: Accept LTPA Token. • Acceptable LTPA versions: WebSphere version 1 and WebSphere version 2 • LTPA key file: upload the LTPA keyfile. • LTPA key file password: specify the password for the LTPA keyfile. • Extract Resource: URL Sent by Client • Authorization: Allow any authenticated client.
Transform	Upload `route.xsl`. See "Sample dynamic routing stylesheet" on page 212. • Input: INPUT • Output: auth
Result	Not applicable.

Table 46. Properties of `worklight-ssl-policy_allowSSLLoginPage`

Property	Value
Direction	Client to Server.
Match	• Type = URL • Pattern = `/(Login|Error)Page\.htm(l)?(\?originalUrl=.*)?`
AAA	• Input: INPUT • Output: NULL Add a new AAA Policy named `AllowSSLLoginPage` with the following configuration: • Method: HTML Form-based Authentication • HTML Form Policy: Create one with the default values, but edit these values: – Use SSL For Login: enabled – SSL Port: port on which the multi-protocol gateway is listening. • Authentication: Pass identity token to authorization phase • Resource extraction: URL sent by client • Authorization: Always allow

Table 46. Properties of `worklight-ssl-policy_allowSSLLoginPage` (continued)

Property	**Value**
Result	Not applicable.

Table 47. Properties of `worklight-ssl-policy_worklightSSLLogin`

Property	**Value**
Direction	Client to Server.
Match	• Boolean Or Combinations: On • Type = URL • Pattern: /worklight/* • Type = URL • Pattern: /j_security_check • Type = URL • Pattern: /applicationcenter/* • Type = URL • Pattern: /appcenterconsole/*
Advanced	"Convert Query Params to XML Action"
AAA	Create a new AAA Policy named `worklightSSLFormLogin` with the following configuration: • Extract Identity: – Method: HTML Form-based Authentication – HTML Form Policy: Select the same policy created in the previous step. • Authentication: – Method: Bind to LDAP server – Enter the HostName, Port(636) – Create an SSL Forward proxy profile with the LDAP server's SSL certificate. – LDAP Bind DN, in this case would be: `cn=admin,dc=worklight,dc=com` – Enter the LDAP Bind Password. – LDAP Prefix : `uid=` – LDAP Suffix : `ou=people,dc=worklight,dc=com` • Resource extraction: URL sent by client • Authorization: Allow any authenticated client. • Post Processing: Generate LTPA Token -> on. – LTPA Token Version: WebSphere version 2 – LTPA Key File: Select the ltpa key file – LTPA key file password: Specify the password for the ltpa keyfile.

Table 47. Properties of `worklight-ssl-policy_worklightSSLLogin` (continued)

Property	Value
Transform	Upload `route.xsl` file. See "Sample dynamic routing stylesheet." • Input: INPUT • Output: auto
Result	Not applicable.

Results

The different pieces of the topology are now configured and provide a seamless SSO experience for mobile applications as well as for Application Center.

Sample dynamic routing stylesheet

You can use this sample stylesheet to handle the dynamic routing of requests between IBM Worklight and Application Center. You refer to the stylesheet when you create rules to define a form-based authentication policy that generates and verifies LTPA tokens.

You provide a custom dynamic routing stylesheet when you define rule `worklight-ssl-policy_verifyLTPA` (see "Integrating IBM WebSphere DataPower with a cluster of Worklight Servers" on page 201, Table 45 on page 209), and when you define rule `worklight-ssl-policy_worklightSSLLogin` (see "Integrating IBM WebSphere DataPower with a cluster of Worklight Servers" on page 201, Table 47 on page 211).

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:dp="http://www.datapower.com/extensions"
  xmlns:dpconfig="http://www.datapower.com/param/config"
  xmlns:re="http://exslt.org/regular-expressions"
  extension-element-prefixes="dp re dpconfig"
  exclude-result-prefixes="dp">

  <xsl:param name="dpconfig:worklightBackend"/>
  <xsl:param name="dpconfig:applicationcenterBackend"/>
  <xsl:template match="/">

    <xsl:variable name="worklight" select="'worklight'"/>
    <xsl:variable name="applicationcenter" select="'applicationcenter'"/>
    <xsl:variable name="appcenterconsole" select="'appcenterconsole'"/>

    <xsl:variable name="worklightBackend" select="$dpconfig:worklightBackend"/>
    <xsl:variable name="applicationcenterBackend" select="$dpconfig:applicationcenterBackend"/>

    <xsl:variable name="incomingURI" select="dp:variable('var://service/URI')"/>
    <xsl:variable name="httpContentType" select="dp:http-request-header('Content-Type')"/>
    <xsl:variable name="accessControlRequestHeaders" select="dp:http-request-header('Access-Control-
    <xsl:variable name="accessControlRequestMethod" select="dp:http-request-header('Access-Control-R

    <xsl:choose>
      <!-- set the backend server if the url is /worklight -->
      <xsl:when test="contains(dp:variable('var://service/URI'), $worklight)">
        <dp:set-http-request-header name="'Content-Type'" value="$httpContentType"/>
        <dp:set-variable name="'var://service/routing-url'" value="$worklightBackend"/>
        <dp:set-variable name="'var://service/URI'" value="$incomingURI"/>
      </xsl:when>


      <xsl:when test="contains(dp:variable('var://service/URI'), $applicationcenter) or contains(dp:v
        <dp:set-http-request-header name="'Content-Type'" value="$httpContentType"/>
        <dp:set-http-request-header name="'Access-Control-Request-Headers'" value="$accessControlR
        <dp:set-http-request-header name="'Access-Control-Request-Method'" value="$accessControlRe
        <dp:set-variable name="'var://service/routing-url'" value="$applicationcenterBackend"/>
        <dp:set-variable name="'var://service/URI'" value="$incomingURI"/>
      </xsl:when>

      <xsl:when test="contains(dp:variable('var://service/URI'), 'j_security_check')">
        <dp:set-http-request-header name="'Content-Type'" value="$httpContentType"/>
        <dp:set-variable name="'var://service/routing-url'" value="$applicationcenterBackend"/>
        <dp:set-variable name="'var://service/URI'" value="'/appcenterconsole/login/j_security_che
      </xsl:when>

      <xsl:otherwise>
        <xsl:message dp:type="all" dp:priority="error"> No matching url found. </xsl:message>
      </xsl:otherwise>
    </xsl:choose>

    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

Installing and configuring IBM SmartCloud Analytics Embedded

To run the analytics features, you must install IBM SmartCloud Analytics Embedded.

About this task

The operational analytics feature requires installation of IBM SmartCloud Analytics Embedded, 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:
- Supported operating systems:
  - AIX V6.1 and V7.1 on ppc 64-bit
  - Red Hat Enterprise Linux (RHEL) 6 Server editions on x86-64
  - Red Hat Enterprise Linux (RHEL) 5 Update 6 Advanced Platform on v86-64
  - SUSE Linux Enterprise Server (SLES) 10 and 11 on x86-64
- 200 MB of disk space is required for the installation.
- 8 GB of RAM is required.
- 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/analytics_[OS].zip` (or your installation location, if different)
- For Windows systems: `c:\Program Files (x86)\IBM\Worklight\Analytics\analytics_[OS].zip` (or your installation location, if different)

Procedure

1. Copy the `analytics_[OS].zip` file to the system that you designated as the host for IBM SmartCloud Analytics Embedded.
2. Extract the `analytics_[OS].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 IBM SmartCloud Analytics Embedded. 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 a cluster name for the search node. If none is specified, the default name of `WLCLUSTER` is used.
   d. Specify the location to install the analytics. If none is specified, the default location of `/opt/IBM/analytics` is used.
   e. Specify the location to store data. If none is specified, the default location `<INSTALL_LOCATION>/data` is used. The console installation does not ask for the location to store data.
   f. Specify the location to store logs. If none is specified, the default location of `<INSTALL_LOCATION>/logs` is used. The console installation does not ask for the location to store logs.
   g. Specify an available port for analytics. If none is specified, the default port 80 is used. The search installation does not ask for port information.
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 `<INSTALL_LOCATION>/iwap.sh start`.

Results

You have installed IBM SmartCloud Analytics Embedded. During the installation this summary appears, providing instructions about how to start, stop, or uninstall IBM SmartCloud Analytics Embedded:

```
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.
  <INSTALL_LOCATION>/iwap.sh start
  <INSTALL_LOCATION>/iwap.sh stop
  <INSTALL_LOCATION>/iwap.sh 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
```

What to do next

Configure the Worklight Server to use analytics by following the steps in "Configuring Worklight Server for analytics."

Related concepts:

Operational analytics
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 SmartCloud Analytics Embedded.

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 27.

About this task

To enable the Worklight Server WAR to forward data to IBM SmartCloud Analytics Embedded, 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 IBM SmartCloud Analytics Embedded, modify the JNDI configuration by completing these steps.

Procedure

1. Find and modify the URL that you noted during the IBM SmartCloud Analytics Embedded 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 IBM SmartCloud Analytics Embedded 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 SmartCloud Analytics Embedded`*`]`. For more information about JNDI properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784.

   When this WAR is installed and the JNDI properties are properly configured, data flows to IBM SmartCloud Analytics Embedded. 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 IBM SmartCloud Analytics Embedded. The user interface for the Worklight Console Analytics tab is rendered within an IFRAME that communicates directly with IBM SmartCloud Analytics Embedded. The browser from which the analytics information is accessed must have direct connectivity to IBM SmartCloud Analytics Embedded, 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

Enable the Analytics optional feature by following the steps in “Enabling analytics” on page 950. For more information about operational analytics, see “Analytics” on page 938.

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
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 the installation failure, you can use the manual installation instructions to investigate the problem more thoroughly. See "Deploying a project WAR file and configuring the application server manually" on page 767.

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

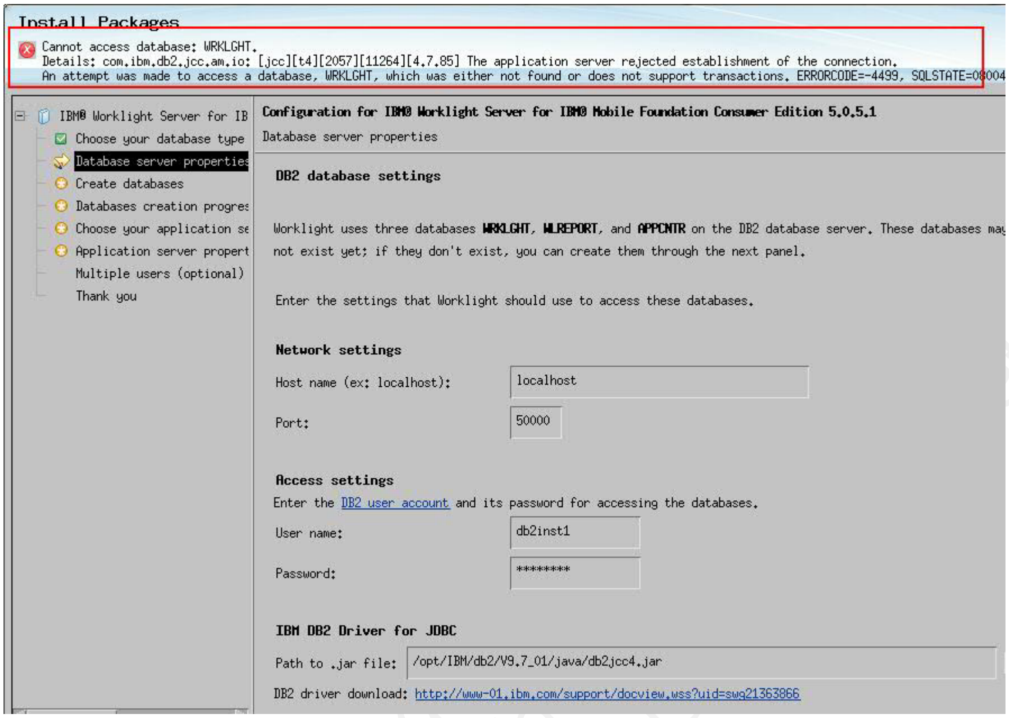
1. Wait a few minutes for the current DB2 database connections to close, and then click **Back** followed by **Next** to check if the issue is solved.
2. If the problem persists, contact your database administrator to solve the database connection issue that is documented on the following web page: http://pic.dhe.ibm.com/infocenter/db2luw/v10r1/index.jsp?topic=%2Fcom.ibm.db2.luw.messages.sql.doc%2Fdoc%2Fmsql01035n.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:


Install Packages
IBM® Worklight Server for IB
Choose your database type
Database server properties
Create databases
Databases creation progres
Choose your application se
Application server propert
Multiple users (optional)
Thank you

```
Cannot access database: WRKLGHT.
Details: com.ibm.db2.jcc.am.io: [jcc][t4][2057][11264][4.7.85] The application server rejected establishment of the connection.
An attempt was made to access a database, WRKLGHT, which was either not found or does not support transactions. ERRORCODE=-4499, SQLSTATE=08004
```

Configuration for IBM® Worklight Server for IBM® Mobile Foundation Consumer Edition 5.0.5.1

Database server properties

DB2 database settings

Worklight uses three databases **WRKLGHT**, **WLREPORT**, and **APPCNTR** on the DB2 database server. These databases may not exist yet: if they don't exist, you can create them through the next panel.

Enter the settings that Worklight should use to access these databases.

Network settings

Host name (ex: localhost): localhost

Port: 50000

Access settings

Enter the DB2 user account and its password for accessing the databases.

User name: db2inst1

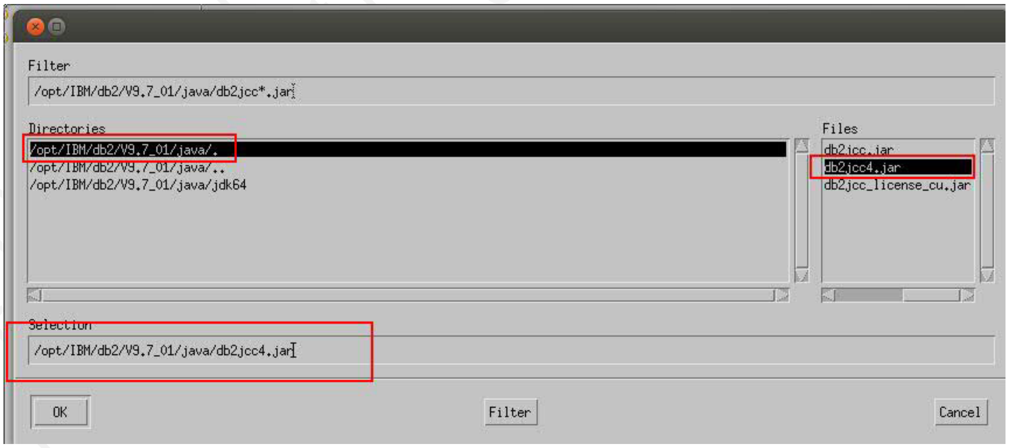
Password: ********

IBM DB2 Driver for JDBC

Path to .jar file: /opt/IBM/db2/V9.7_01/java/db2jcc4.jar

DB2 driver download: http://www-01.ibm.com/support/docview.wss?uid=swg21363866

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:



Filter: /opt/IBM/db2/V9.7_01/java/db2jcc*.jar

Directories:

```
/opt/IBM/db2/V9.7_01/java/.
/opt/IBM/db2/V9.7_01/java/..
/opt/IBM/db2/V9.7_01/java/jdk64
```

Files:

```
db2jcc.jar
db2jcc4.jar
db2jcc_license_cu.jar
```

Selection: /opt/IBM/db2/V9.7_01/java/db2jcc4.jar

Procedure

1. If you receive this error, verify the current DB2 JBDC driver.
2. Newer fix packs of the DB2 JDBC driver may solve the issue. These fix pack drivers are available from DB2 JDBC Driver Versions.

Troubleshooting a Worklight Server upgrade with Derby as the database

If Application Center is installed and uses Apache Derby as a database, stop the application server that runs the application before you run IBM Installation Manager to upgrade a Worklight Server installation.

About this task

During an upgrade of Worklight Server, if Application Center is installed, the installer migrates the database that is used by Application Center. When Apache Derby is the database, this operation can fail if the application server that runs Application Center is not stopped.

The symptom of this problem is that the upgrade fails and the log file contains the error message `Another instance of Derby may have already booted the database`.

Procedure

Before you run IBM Installation Manager to upgrade an installation of Worklight Server and Application Center, stop the application server that runs the Application Center application.

Troubleshooting failure to authenticate to Application Center and applications that use the basic registry element

Authentication fails when attempting to log in to the Application Center and other applications that run on WebSphere Application Server Liberty profile and use the `basicRegistry` element.

About this task

When IBM Worklight is installed with Application Center on WebSphere Application Server Liberty profile, it adds a `basicRegistry` element in the `server.xml` file of the Liberty server instance, with demo users, even if a `basicRegistry` element already exists. Authentication into the Application Center and other applications that use users from the basic registry no longer works. For example, after an attempt to log in to the Application Center, the following error message is displayed:

```
Error 404: java.io.FileNotFoundException: SRVE0190E: File not found: /j_security_check
```

The liberty server log file contains the following error message:

```
[ERROR] CWWKS3006E: A configuration exception has occurred. There are multiple available UserRegis
```

When IBM Worklight is uninstalled, the basic registry that was created during the installation by the Worklight Server installer is removed from the `server.xml file`, even if other users have been added to that basic registry. If other applications than Application Center use the basic registry, authentication on these applications is no longer possible.

Procedure

1. Move the content of the basic registry that was created by IBM Installation Manager in the initial basic registry element. For an installation that is not for test purposes only, do not copy the users `demo` and `appcenteradmin`, and remove them from the `appcentergroup`. Remove the following code from the `server.xml` file:

```
<!-- Declare the user registry for the Application Center. -->
<basicRegistry id="applicationcenter-registry" realm="ApplicationCenter">
  <!-- The users defined here are members of group "appcentergroup", thus have role "appcenter
  <user name="appcenteradmin" password="admin"/>
  <user name="demo" password="demo"/>
  <group name="appcentergroup">
```

```
    <member name="appcenteradmin"/>
    <member name="demo"/>
  </group>
</basicRegistry>
```

2. When you uninstall IBM Worklight, the uninstaller of Worklight Server creates a backup of the `server.xml` file under the name `server.xml.saved2`. Open the `server.xml.saved2` file, and copy the `basicRegistry` element back in the `server.xml` file. Remove the users and groups that were only needed by the Application Center.

Mobile testing for IBM Worklight

Contents

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.

Note: Testing web applications is not supported in IBM Mobile Test Workbench for Worklight. Some topics in this section mention support for web apps, but that support does not apply to IBM Mobile Test Workbench for Worklight.

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, Linux, or a Macintosh 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.

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.

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.

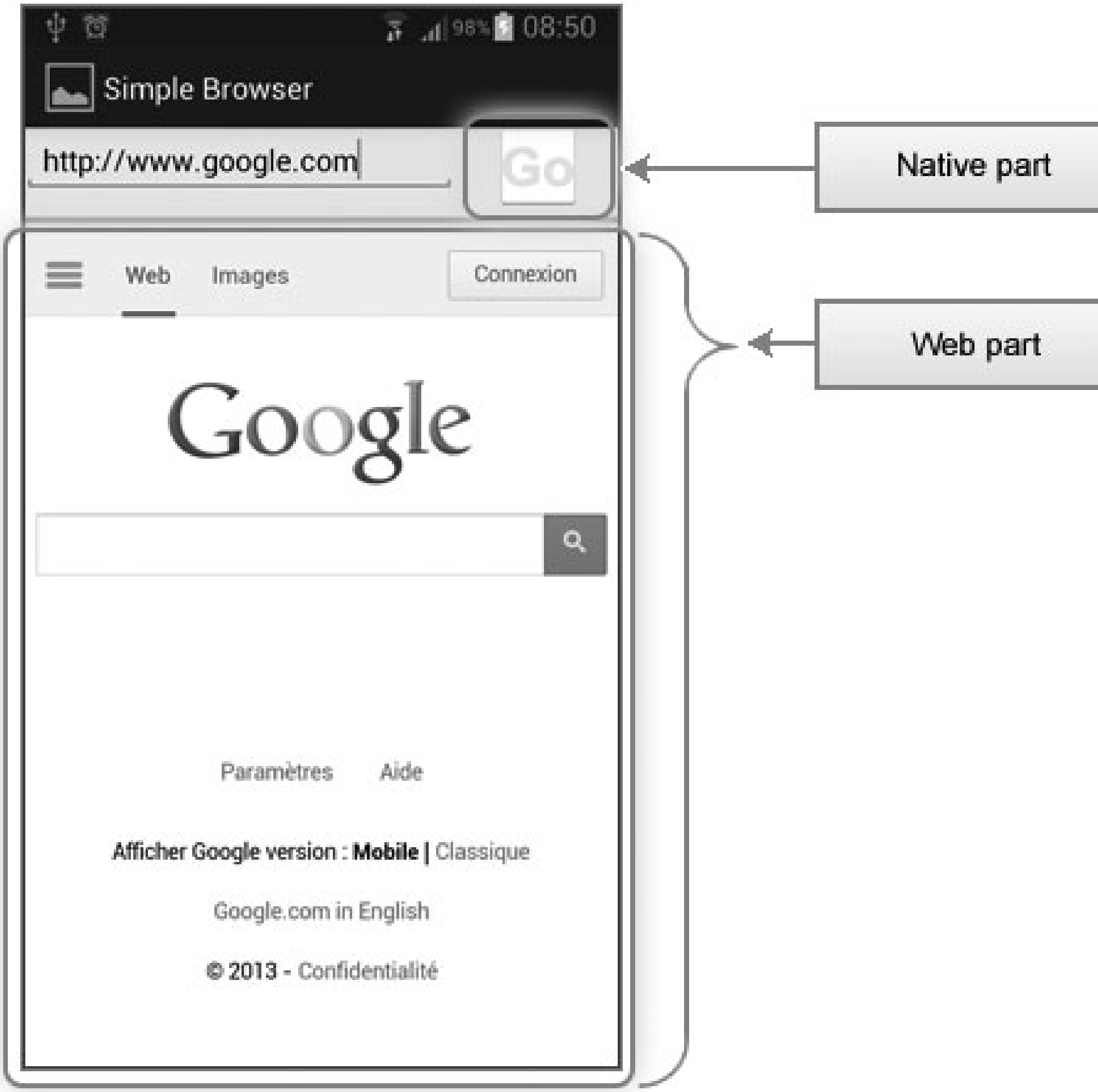
Support for testing native and hybrid applications

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.



Related information:
Android testing overview
iOS testing overview

Creating a Test Workbench 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

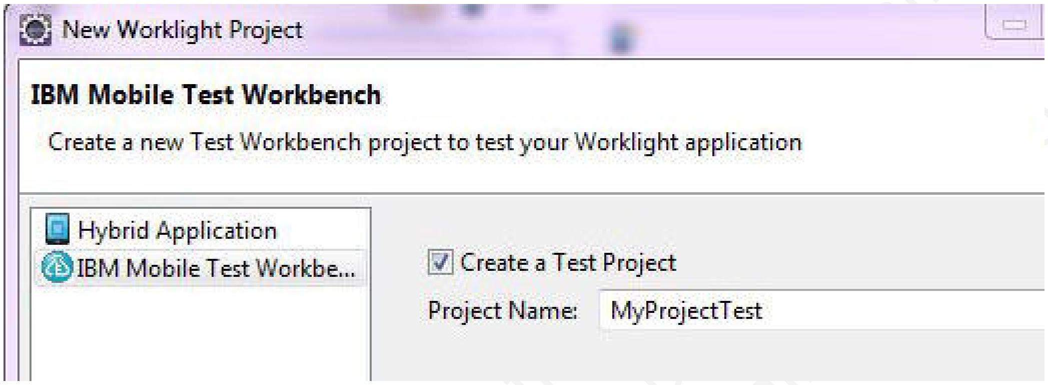
You can easily create a test project by selecting **File** > **New** > **Test Workbench Project** within Eclipse as described in Creating a Test Workbench project, or you can create a test project at the same time as you create your IBM Worklight project.

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 product.

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.



New Worklight Project

IBM Mobile Test Workbench

Create a new Test Workbench project to test your Worklight application

☑ Create a Test Project

Project Name: MyProjectTest

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

You must prepare your application for testing by building the environment, and by running your app on the Worklight Development Server. To do so, complete the following steps:

1.
   - For Android apps:
     a. Create the native Android project in Worklight Studio by right-clicking your application, and clicking **Run As** > **Build Android Environment**.
     b. Create the application binary file (the APK file) by using the Android tools.

   - For iOS apps: Create the Xcode project in Worklight Studio by right-clicking your application, and clicking **Run As** > **Build IPhone Environment**. The appropriate certificate is specified in the Xcode project, in case you want to test your app on a real device.
2. Perform a build and deployment action on your project by right-clicking the project name, and clicking **Run As** > **Run on Worklight Development Server** to make the iOS `.ipa` file or the Android `.apk` file available, and to update the Android project.
3. (For Android only) Compile the APK by right-clicking the name of the automatically generated Android project, and clicking **Run As** > **Android Studio project**.

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.
- 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.

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**.

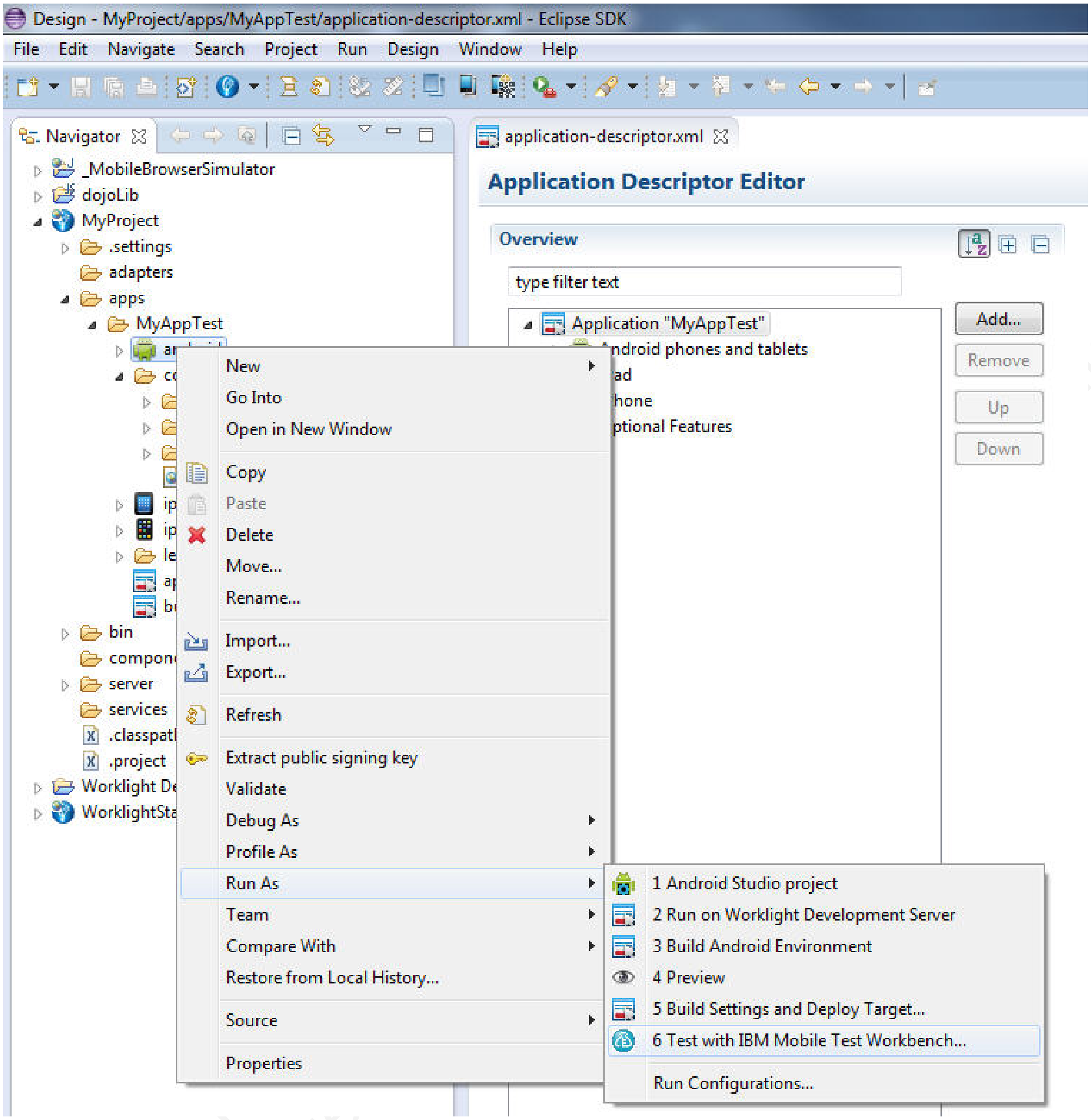

Design - MyProject/apps/MyAppTest/application-descriptor.xml - Eclipse SDK
File Edit Navigate Search Project Run Design Window Help
Navigator
_MobileBrowserSimulator
dojoLib
MyProject
.settings
adapters
apps
MyAppTest
bin
server
services
.project
application-descriptor.xml
Application Descriptor Editor
Overview
type filter text
Application "MyAppTest"
ndroid phones and tablets
ptional Features
Add...
Remove
Up
Down
New
Go Into
Open in New Window
Copy
Paste
Delete
Move...
Rename...
Import...
Export...
Refresh
Extract public signing key
Validate
Debug As
Profile As
Run As
Team
Compare With
Restore from Local History...
Source
Properties
1 Android Studio project
2 Run on Worklight Development Server
3 Build Android Environment
4 Preview
5 Build Settings and Deploy Target...
6 Test with IBM Mobile Test Workbench...
Run Configurations...

Figure 2. Testing with IBM Mobile Test Workbench from Worklight Studio

- For Android applications, this action places the `.apk` file 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 iOS applications on the iOS Simulator.

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."

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

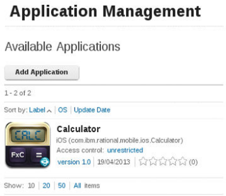
Testing iOs applications requires that each application is instrumented before you can record tests on it with IBM Mobile Test Workbench for Worklight. Applications must be instrumented before they are published in the Application Center catalog.

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.



Application Management

Available Applications

Add Application

1 - 2 of 2

Sort by: Label | OS | Update Date

Calculator
iOS (com.ibm.rational.mobile.ios.Calculator)
Access control: unrestricted
version 1.0 | 19/04/2013 | ☆☆☆☆☆ (0)

Show: 10 | 20 | 50 | All items

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.

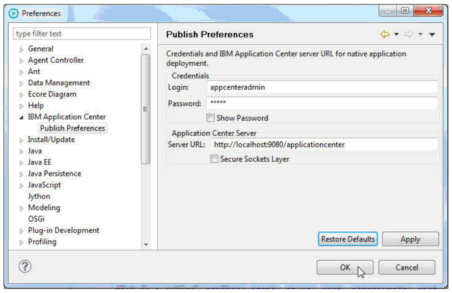
About this task

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.



Publish Preferences

Credentials and IBM Application Center server URL for native application deployment.

Credentials

Login: appcenteradmin

Password: *****

□ Show Password

Application Center Server

Server URL: http://localhost:9080/applicationcenter

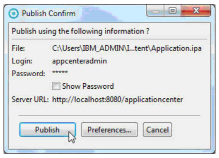
□ Secure Sockets Layer

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.



Publish using the following information ?

File: C:\Users\IBM_ADMIN\I...tent\Application.ipa

Login: appcenteradmin

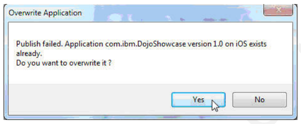
Password: *****

□ Show Password

Server URL: http://localhost:8080/applicationcenter

Figure 5. Options to publish the application or change the publication preferences

If the application already exists, publication will fail.



Publish failed. Application com.ibm.DojoShowcase version 1.0 on iOS exists already.
Do you want to overwrite it ?

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.

Creating a Test Workbench project

The tests that you create, and the assets associated with the test, reside in a test project. You can create the project separately, or you can simply record a test, which automatically creates a project named `testproj`.

Procedure

1. Select **File** > **New** > **Test Workbench Project**. The Create a Test Workbench Project window opens.
2. In the **Project Name** field, type a name for the project. If you plan to collect response time breakdown data, do not use a project name that contains spaces.
3. Select **Use default location**.
4. Optional: Click **Next** and select the folders to create in the new project. These folders organize your files by asset (`Tests`, , `Results`, and so on).
5. Click **Finish**.

   After you click finish, you are prompted to record a test. You can create a test from a new recording or from an existing recording, or just click **Cancel** to create a test project without recording a test.

Getting started with mobile testing

Look here to find an overview of the steps to start testing Android and iOS applications.

Android testing overview

With the mobile test client for Android you can test native Android applications, web applications, and hybrid applications from your Android device and from Android emulators.

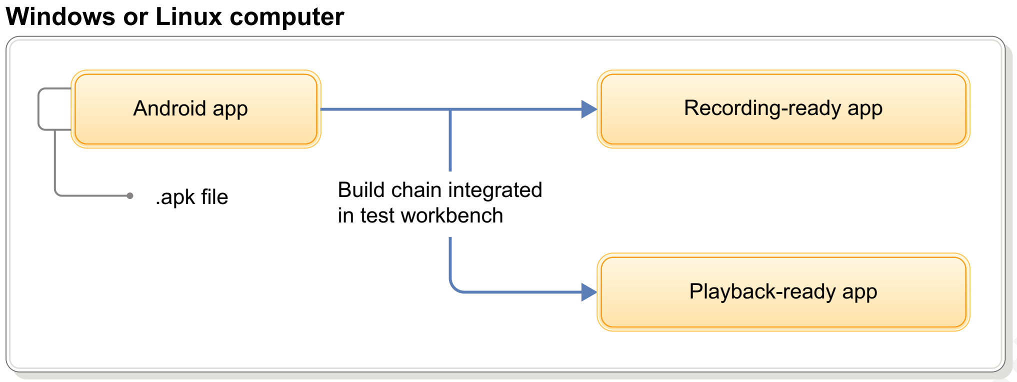
The mobile test client for Android works in conjunction with the test workbench. From any of the clients you can connect to the test workbench, record and run tests, and view reports. From the Android client, you can also upload applications to the test workbench, where the applications are instrumented and recompiled into two new apps: a recording-ready application and a playback-ready application.

The following figure shows the native Android mobile test client:

How the test workbench tests Android applications

The mobile test client for Android is a native Android application that runs on Android devices and emulators. Each Android application is packaged into a single `.apk` file that includes the applications code and other resource files. The `.apk` file is in a compressed format, similar to a `zip` file or a `war` file.

For Android, the build and compile process takes place in the test workbench on a Windows or Linux computer.


Windows or Linux computer
Android app
.apk file
Build chain integrated in test workbench
Recording-ready app
Playback-ready app

Before you can test a mobile 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 record a test, the Android application (the `.apk` file) is recompiled into a *recording-ready app* that has been heavily instrumented to capture user actions. Because Android does not allow two versions of an application to be installed at the same time, the test workbench uninstalls the original application and replaces it with the recording-ready app. When you play back a test, the test workbench uninstalls the recording-ready app and replaces it with a *playback-ready app*, a version of the original application that has been signed with a test workbench certificate.

Note: There is also another version of the app, the *Tester app*. This app contains the runtime code that is needed to replay a test. This app will not be 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.

You can simplify this process of installing and uninstalling versions of the Android app by choosing **Playback on instrumented** from the Settings page on your Android device or emulator. This lets you play back a test using the more heavily instrumented recording version of the app, rather than the lighter weight playback version of the app. This is at the expense, however, of slower playback speed and greater memory consumption.

Passive mode

When you are ready to run your tests, you can run them on the device or in the test workbench. To give control to the test workbench, simply tap **Enter Passive Mode** from the mobile test client.

iOS testing overview

With the mobile test client for iOS you can test native iOS applications, web applications, and hybrid applications. From your iOS device and from the iOS Simulator you can connect to the test workbench, record and run tests, and view reports.

Testing on an iOS device

You can test native, hybrid, and web applications on an iOS device using the browser-based client for iOS. This client is a web application that runs in the Safari or Chrome browser on your device. To run the client, simply type the Workbench URL in the following format in a browser on your device:

Format: `http://`*`Workbench_URL`*`:`*`port`*`/mobile`

Example: `http://9.11.22.333:7878/mobile`

The following figure shows the mobile test client in a browser on an iOS device:

Here are some important details about the browser-based client:

- The browser-based client can detect API information, device orientation, locale, and device type from the device.
- The browser-based client does not detect that a native application is installed on the device. If the user starts a recording or playback before installing the native application, it will result in an `Invalid URL` message. This means that the instrumented native application cannot be found on the device. All native applications are identified by their URLScheme.
- Whenever an application under test is started. it opens as a URL in a new browser tab. The blank screen that is shown is actually a new tab and is shown temporarily before the launch.

  Note: When using the browser-based client, be sure to clear the browser cache before connecting to another test workbench.

Testing with the iOS Simulator

You can also do your testing using the Safari browser in the iOS Simulator, or you can use the native mobile test client in the Simulator after following the installation instructions at Installing the mobile test client on the iOS Simulator. Chrome is not currently available on the iOS Simulator.

Enabling pop-up windows

When using the mobile test client on an iOS device, be sure to enable pop-up windows. Pop-up windows must be allowed for the Workbench IP address. Safari silently disables pop-up windows, while Google Chrome prompts you to allow pop up windows. For every device-specific operation (install, record, playback), Safari prompts with a pop-up confirmation. There are fewer messages with Chrome.

Passive mode

When you are ready to run your tests, you can run them on the device or in the test workbench. To give control to the test workbench, simply tap **Enter Passive Mode** on the mobile test client.

Passive mode behavior is browser-specific. While Chrome operates in passive mode as expected. Safari is more restrictive and displays multiple passive mode confirmation dialog boxes.

Differentiating among multiple devices

When you connect a particular device, such as an iPhone, to the test workbench, you can view the connection in the Mobile Devices editor in the test workbench.

To differentiate among multiple iPhones that are connected to the test workbench, you can change the Device Description.

1. On the device, open Safari or Chrome and browse to the Workbench URL.

   This opens the mobile test client.
2. Tap **About** .
3. Type the new label in the Device Description field.
4. Click **Register**.

You can see the result in the figures below.

Table 2.

Before	**After**

IBM Rational Test Workbench Mobile Web Recorder

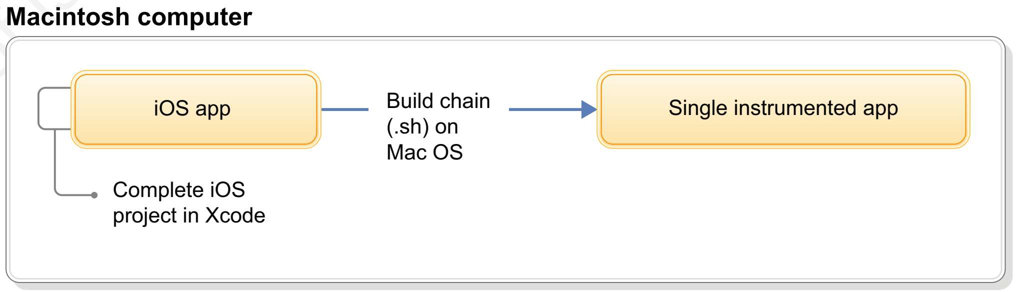
The mobile web recorder is an iOS application that is required to record and play back user actions of a web application under test. It is invoked automatically during the recording and playback process. You can download the mobile web recorder from the Apple App Store. When you are done testing, you can stop the mobile web recorder using the standard Apple Activity Monitor.

How the test workbench tests iOS applications

Before you can test a mobile 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.

A native 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 following figure shows the build chain for a native iOS app on a Macintosh computer:



Related reference:

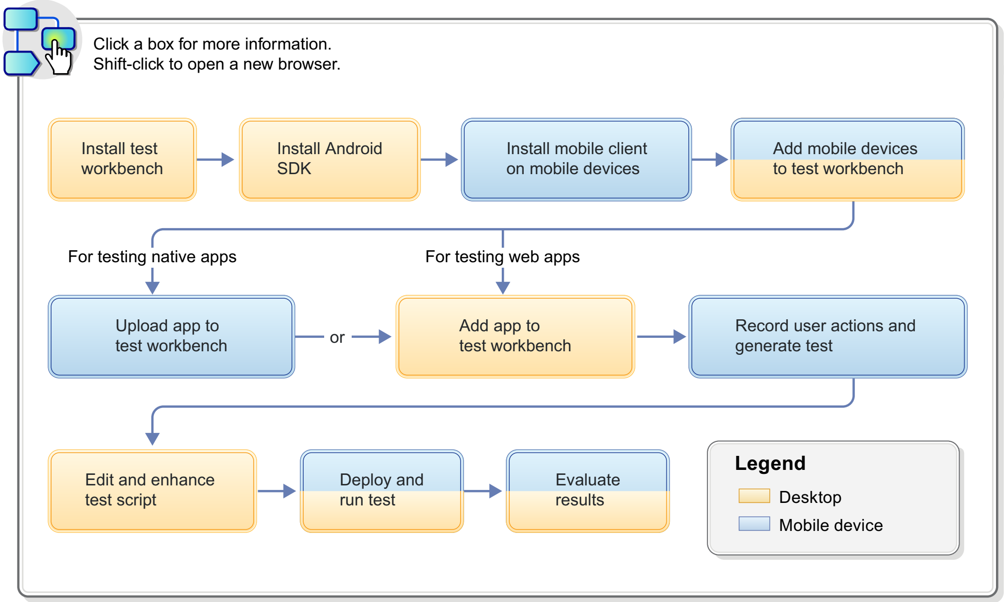
Software and hardware requirements
Before you install the mobile test client and start working with the test workbench, be sure to understand the following requirements.

Getting started with Android testing

Use this topic to help you get started with your testing of applications that run on Android devices.

About this task

The following diagram shows is a high-level overview of mobile testing for Android:


Click a box for more information.
Shift-click to open a new browser.
Install test workbench
Install Android SDK
Install mobile client on mobile devices
Add mobile devices to test workbench
For testing native apps
For testing web apps
Upload app to test workbench
or
Add app to test workbench
Record user actions and generate test
Edit and enhance test script
Deploy and run test
Evaluate results
Legend
Desktop
Mobile device

Procedure

1. Set up your mobile test environment.
   a. Install IBM Mobile Test Workbench for Worklight and ensure that IBM Mobile Test Workbench for Worklight and Rational Test Workbench Extension for Mobile are selected.
   b. Install the Android SDK on the same computer that the test workbench is installed on.

      You do not need to install the full Android Developer Tools (ADT) bundle, but be sure to install Android SDK Tools, Android SDK Platform-tools, and Android SDK Build-tools, if these are not already installed. The current download page of the Android SDK is at Get the Android SDK but the location could change in the future.

      Note: Running the Android SDK Manager and using the Android emulators requires the Oracle Java Developer Kit (JDK). Be sure to also add the path to the Oracle JDK to your system's PATH environment variable.
   c. Run the SDK manager at least once before running the test workbench to get the API-level components.
   d. Set the preferences in the mobile application builder path so that they point to the directory where the Android SDK is installed. See Mobile application builders.
   e. Install the mobile test client on one or more Android devices or on an emulator. Ensure that the mobile devices can connect to the test workbench over WiFi or a cellular plan.

f. Add your mobile devices to the test workbench.

2. Prepare your mobile application for testing. You can do this in one of the following two ways:
   - From your mobile device, upload the native or hybrid mobile application to the test workbench.
   - From the test workbench, add the application to the test workbench.
3. From the test workbench, create a test project by clicking **File** > **New** > **Test Workbench Project**.
4. Verify that the application is visible in the Test Navigator view in the test workbench.
5. Create a test by recording gestures and user interactions on the mobile device.

   You can initiate the recording on the device or from the Eclipse client. Actions are represented in natural language, which allows you to document and reproduce the test manually.
6. Examine and enhance the test script recording as needed. Here are some of the things you can do:
   - Explore controls and context menus to customize the script to your needs.
   - Add application stubs to simulate the behavior of actual objects that interact with the application under test.
   - Add verification points to verify that an expected value or behavior is returned during a test run.
   - Add variables to the test script.
7. Deploy and run the test, either from the mobile device or from the test workbench.
8. Evaluate the test results.

Related concepts:

Installing Rational Test Workbench
The information in this file applies to IBM Rational Test Workbench, version 8.5. Rational Test Workbench delivers end-to-end functional, regression, load, and integration testing to address the quality challenges of highly complex and integrated applications.

Installing the mobile test client
This section contains instructions for installing the IBM Rational Test Workbench Mobile Client.

"Editing Mobile tests" on page 34
With the test editor, you can view and customize a mobile test imported in the test workbench or uploaded from a mobile device or simulator after recording.

Related tasks:

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 mobile test client to connect to the test workbench.

"Creating a Test Workbench project" on page 10
The tests that you create, and the assets associated with the test, reside in a test project. You can create the project separately, or you can simply record a test, which automatically creates a project named `testproj`.

"Recording tests from the Android mobile test client" on page 31
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

"Recording tests from the test workbench" on page 34
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.

"Running tests from an Android mobile test client" on page 44
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.

"Running tests from the test workbench" on page 46
After generating the test from a recording, you can edit a test according to your requirements and play it back from the workbench on your mobile device/simulator. This means that the playback is controlled from the workbench and not from the simulator or mobile device. The result of the tests can be viewed from both the test workbench and the mobile test client.

"Evaluating results" on page 50
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.

Getting started with testing on iOS devices

You can test native, hybrid, and web applications on an iOS device using the browser-based client for iOS. This client is a web application that runs in the Safari or Chrome browser on your device.

Before you begin

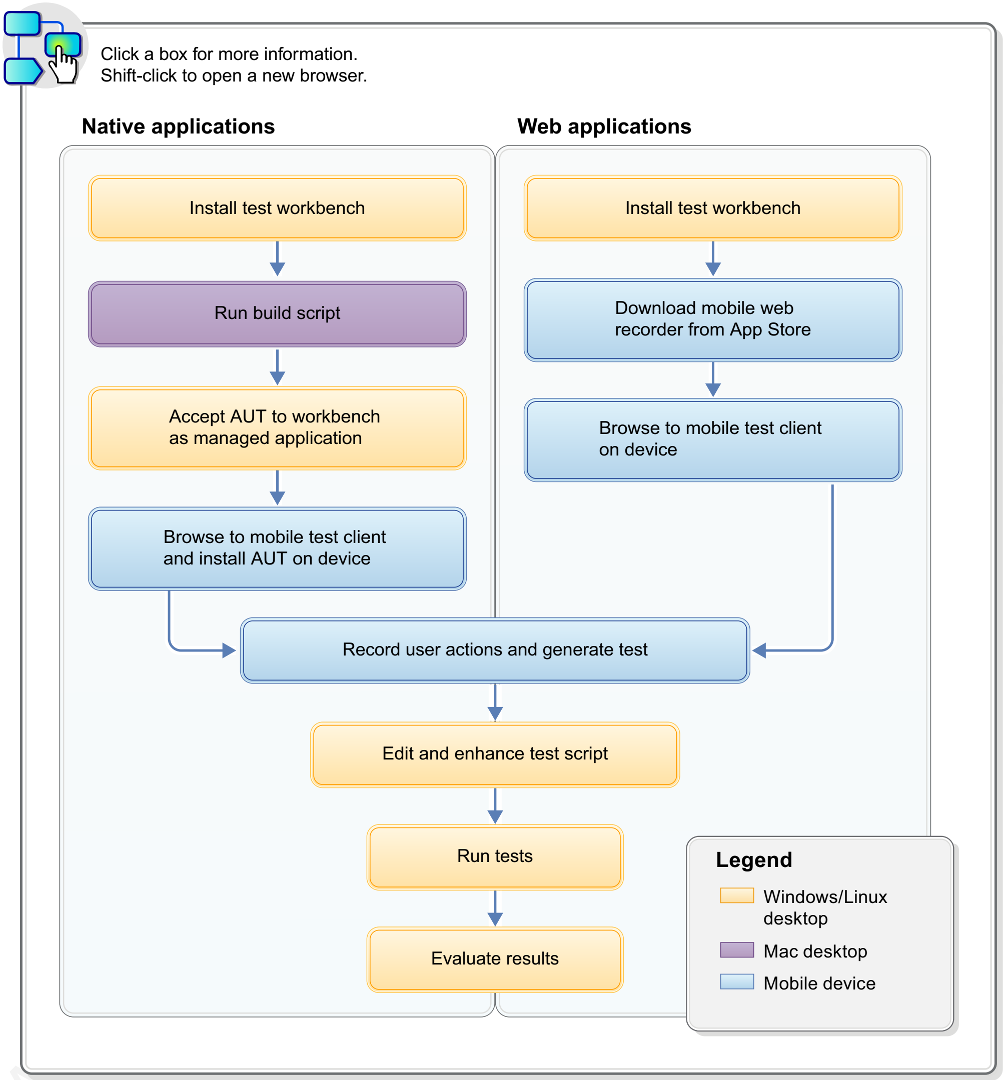
Note the following requirements:

- You will need an Apple Developer or Enterprise License with a provisioning profile for each mobile device.
- To record and play back tests of a native or hybrid iOS application, you will need a managed version of the application under test (AUT). If this is not available, you will need to instrument the AUT on a Macintosh computer that contains the Xcode source project for the AUT. A Macintosh computer is not required for testing web applications.

Note: Xcode is the Apple integrated development environment (IDE) used for developing Macintosh and iOS applications. You can download Apple Xcode from the Xcode Dowloads and Resources web site or from the Apple App Store on a Macintosh computer.

About this task

The following diagram shows a high-level overview of mobile testing for iOS applications:


Click a box for more information.
Shift-click to open a new browser.
Native applications
Web applications
Install test workbench
Run build script
Accept AUT to workbench as managed application
Browse to mobile test client and install AUT on device
Install test workbench
Download mobile web recorder from App Store
Browse to mobile test client on device
Record user actions and generate test
Edit and enhance test script
Run tests
Evaluate results
Legend
Windows/Linux desktop
Mac desktop
Mobile device

Note: With IBM Mobile Test Workbench for Worklight you can also perform any of the desktop steps on a Macintosh computer.

Procedure

1. Install IBM Mobile Test Workbench for Worklight on a Windows or Linux computer and ensure that IBM Mobile Test Workbench for Worklight and Extension for Mobile are selected. For details, see Installing the product software.
2. From the test workbench, create a test project by clicking **File** > **New** > **Test Workbench Project**.
3. If you need to build and instrument a native or hybrid iOS app for testing, proceed as follows:
   a. Download the build archive on to a Macintosh computer.
   b. Run `rtwBuildXcode.sh` to instrument the AUT and send it to the test workbench as an incoming application.

   For details, see "Instrumenting iOS applications on an iOS device" on page 27.

4. If you are testing web applications, download the IBM Rational Test Workbench Mobile Web Recorder from the Apple App Store on to your mobile device.

   The mobile web recorder is required to record and play back user actions of a web application under test. It is invoked automatically during the recording and playback process.
5. In the test workbench add the application under test as a managed, instrumented application. For details, see “Importing applications to test in the workbench” on page 23 and “Adding web applications to test workbench” on page 25.
6. For testing native and hybrid applications, open the mobile test client in a browser on your device and add the managed, instrumented AUT on the device.
7. Create a test by recording gestures and interactions on the device. Actions are represented in natural language, which allows you to document and reproduce the test manually.
8. Examine and enhance the test script recording as needed. Here is a partial list of what you can do:
   - Explore controls and context menus to customize the script to your needs.
   - Add stubs to simulate the behavior of actual objects that interact with the AUT.
   - Add verification points to verify that an expected value or behavior is returned during a test run.
   - Add variables to the test script.
9. Run the test on the iOS device or in the test workbench.
10. Evaluate the test results.

Related concepts:

“Editing Mobile tests” on page 34
With the test editor, you can view and customize a mobile test imported in the test workbench or uploaded from a mobile device or simulator after recording.

Related tasks:

“Creating a Test Workbench project” on page 10
The tests that you create, and the assets associated with the test, reside in a test project. You can create the project separately, or you can simply record a test, which automatically creates a project named `testproj`.

“Recording tests from the iOS mobile test client” on page 32
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

“Recording tests from the test workbench” on page 34
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.

“Running tests from the iOS mobile test client” on page 45
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.

"Running tests from the test workbench" on page 46
After generating the test from a recording, you can edit a test according to your requirements and play it back from the workbench on your mobile device/simulator. This means that the playback is controlled from the workbench and not from the simulator or mobile device. The result of the tests can be viewed from both the test workbench and the mobile test client.

"Evaluating results" on page 50
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.

Related reference:

Software and hardware requirements
Before you install the mobile test client and start working with the test workbench, be sure to understand the following requirements.

Getting started with testing on the iOS Simulator

Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

Before you begin

Note:

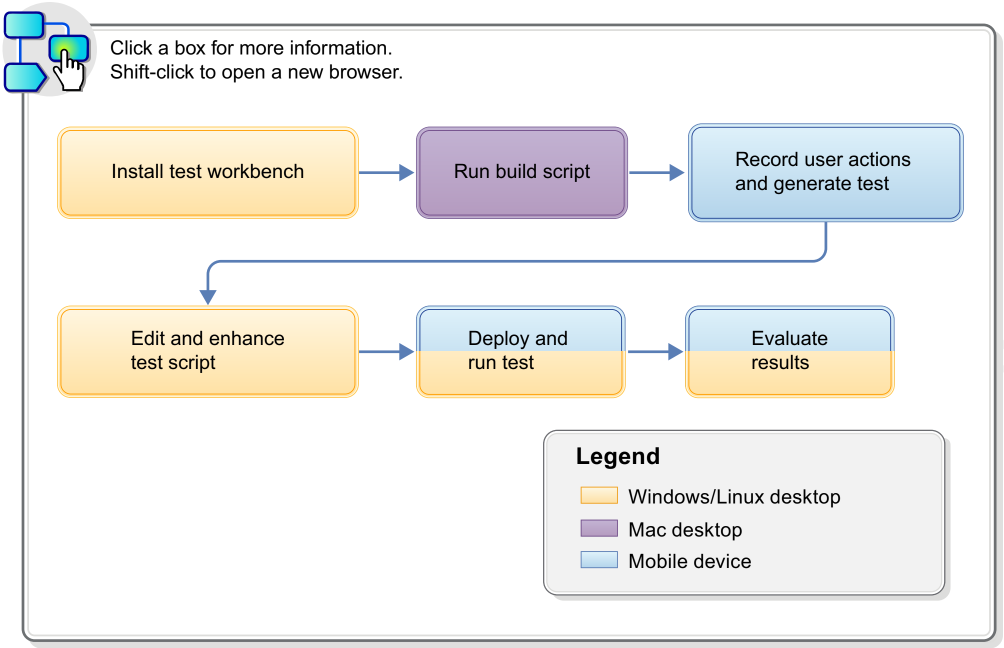
Note the following requirements:

- Testing with the iOS Simulator requires a Macintosh computer and Xcode.
- To record and play back tests of a native or hybrid iOS application, you will need a managed version of the application under test (AUT). If this is not available, you will need to instrument the AUT on a Macintosh computer that contains the Xcode source project for the AUT.
- To test with the native mobile test client in the iOS Simulator, you will first need to install the client. For details, see Installing the mobile test client on the iOS Simulator.

Note: Xcode is the Apple integrated development environment (IDE) used for developing Macintosh and iOS applications. You can download Apple Xcode from the Xcode Dowloads and Resources web site or from the Apple App Store on a Macintosh computer.

About this task

The following diagram shows a high-level overview of mobile testing for iOS applications using the iOS Simulator:


Click a box for more information.
Shift-click to open a new browser.
Install test workbench
Run build script
Record user actions and generate test
Edit and enhance test script
Deploy and run test
Evaluate results
Legend
Windows/Linux desktop
Mac desktop
Mobile device

Note: With IBM Mobile Test Workbench for Worklight you can also perform any of the desktop steps on a Macintosh computer.

Procedure

1. Install IBM Mobile Test Workbench for Worklight on a Windows or Linux computer and ensure that Eclipse ClientIBM Mobile Test Workbench for Worklight and Extension for Mobile are selected. For details, see Installing the product software.
2. From the test workbench, create a test project by clicking **File** > **New** > **Test Workbench Project**.
3. If you need to build and instrument a native iOS app for testing, proceed as follows:
   a. Download the build archive on to a Macintosh computer.
   b. Run `rtwBuildXcode.sh` to instrument the AUT and send it to the test workbench.

   For details, see "Instrumenting iOS applications on the iOS Simulator" on page 29.
4. In the test workbench add the application under test as a managed, instrumented application. For details, see "Importing applications to test in the workbench" on page 23.
5. For testing native applications, open the mobile test client in the iOS Simulator. To run the browser-based client, simply type the Workbench URL in the following format in a browser on your device:

   Format: `http://`*`Workbench_URL`*`:`*`port`*`/mobile`

   Example: `http://192.0.2.24:7878/mobile`
6. Create a test by recording gestures and interactions. Actions are represented in natural language, which allows you to document and reproduce the test manually.
7. Examine and enhance the test script recording as needed. Here are some of the things you can do:

- Explore controls and context menus to customize the script to your needs.
- Add application stubs to simulate the behavior of actual objects that interact with the application under test.
- Add verification points to verify that an expected value or behavior is returned during a test run.
- Add variables to the test script.

8. Run the test in the iOS Simulator or in the test workbench.
9. Evaluate the test results.

Related concepts:

"Editing Mobile tests" on page 34
With the test editor, you can view and customize a mobile test imported in the test workbench or uploaded from a mobile device or simulator after recording.

Related tasks:

"Instrumenting iOS applications on the iOS Simulator" on page 29
Once the mobile test client is installed on the iOS Simulator, you must run a supplied build script on an OS Xsystem to be able to instrument the iOS application under test. The instrumented app can be optionally pushed to the test workbench and installed on the iOS Simulator.

"Creating a Test Workbench project" on page 10
The tests that you create, and the assets associated with the test, reside in a test project. You can create the project separately, or you can simply record a test, which automatically creates a project named `testproj`.

"Recording tests from the iOS mobile test client" on page 32
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

"Recording tests from the test workbench" on page 34
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.

"Running tests from the iOS mobile test client" on page 45
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.

"Running tests from the test workbench" on page 46
After generating the test from a recording, you can edit a test according to your requirements and play it back from the workbench on your mobile device/simulator. This means that the playback is controlled from the workbench and not from the simulator or mobile device. The result of the tests can be viewed from both the test workbench and the mobile test client.

"Evaluating results" on page 50
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.

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 you can 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 import a test suite from a new version of an app, refactoring changes will be applied to the test suite.

You must add web applications in this editor to be able to test them on a device.

Importing applications to test in the workbench

To test an application, you can either use the device to upload the application under test to the Eclipse Client or import the application into the test workbench using the **Mobile applications** editor.

Before you begin

If you are testing Android applications, the Android SDK must be installed on the same computer that the test workbench 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.

About this task

This task applies to Android and iOS applications.

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 applications from the mobile test client" on page 25.

   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**. 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 for test. As a result, a .ma file is created, and the application added to the list of available applications, ready for all the test stages.

   Note: If you import a new version of an application in the **Mobile application** editor, for which test suites had been created from former versions of the app, a dialog box opens and indicates that test suites referencing other versions of the same application have been found. Click **Preview**, this opens a refactoring wizard that displays changes. You can ignore and click **Cancel**, or click **Finish** to start refactoring. Once refactored, the test suites can be used with the new version of the application. If you clicked **Cancel**, you can perform the refactoring process later, using **Import test suite from other version of application** in the **Available test** tab. This button is enabled only if another version of the app is detected in the editor, but is not available for incoming apps, nor for web apps.

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:

"Android testing overview" on page 11
With the mobile test client for Android you can test native Android applications, web applications, and hybrid applications from your Android device and from Android emulators.

"iOS testing overview" on page 12
With the mobile test client for iOS you can test native iOS applications, web applications, and hybrid applications. From your iOS device and from the iOS Simulator you can connect to the test workbench, record and run tests, and view reports.

Related tasks:

"Getting started with Android testing" on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

"Getting started with testing on iOS devices" on page 17
You can test native, hybrid, and web applications on an iOS device using the browser-based client for iOS. This client is a web application that runs in the Safari or Chrome browser on your device.

Adding web applications to test workbench

To test web applications from your Android or iOS devices or emulators, you must first add them to the Rational Test Workbench. When you add the web application to the test workbench, it shows up in the Rational Test Workbench Mobile test client.

Before you begin

The mobile test clients must be connected to test workbench. See Configuring the mobile test client.

Procedure

1. To open the Mobile Applications editor, click the **Display available mobile applications** icon or open an existing application from the Test Navigator view.
2. On the Mobile Applications editor toolbar, click the **Add web application to list** icon .
3. Choose to add a new or an existing application:
   - To add a new web application, complete the following steps:
     - In the **Address** field, type the URL of the web application.
     - In the **App Context** field, type the context root of the web application.
   - To add an existing managed application, complete the following steps:
     - Click **From existing Managed App resource** and click the Browse button.
     - Select a managed application and click **OK**. You must have created a managed application and added it to your project workspace.
4. Optional: In **Application description**, type a description of the web application.
5. Click **Next**. Ensure that the file name is correct and it is associated with an appropriate project.
6. Click **Finish**.
7. Verify that the web application is available on your mobile device.
   a. Open the mobile test client in a browser by typing the workbench URL.
   b. Tap **Manage Web Applications**.

   The web app that you added should now be listed.

What to do next

You can now record a test for the web applications. To record a test on an iOS device, you must first install IBM Rational Test Workbench Mobile Web Recorder from the Apple Store.

Uploading Android applications from the mobile test client

To test mobile apps, you must import them or upload them to . An original package must be uploaded for each application to test and saved in the workbench. The application are also instrumented and recompiled into two new apps: a recording-ready application and a playback-ready application as soon as they are imported or uploaded in the test workbench. 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 if you are testing native or hybrid applications using the mobile test client on Android/devices or iOS simulators.

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 **Processing** tag until the app is fully uploaded and instrumented for testing. The **Test Navigator** displays the application in **Incoming applications**. An original package is created and copied in your workspace to be used in the test.
3. Right-click the incoming application and select **Generate managed application**. In the wizard that opens up, select or create a project or folder to save the .ma resource that contains your original mobile application package. You can change the .ma file name.
4. Click **Finish**. The application is displayed in the **Mobile application** editor and in the **Test Navigator** under **Mobile applications**. The application is ready for all the test stages and it is identified as a managed application on the mobile device.

What to do next

When the application is ready, you can record your test. For more information, see “Recording tests from the Android mobile test client” on page 31

Related tasks:

“Getting started with Android testing” on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

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 iOS applications on an iOS device

This topic describes the steps for preparing a native iOS application for testing on an iOS device. You will need to perform steps in the test workbench, on a Macintosh computer, and on an iOS device.

Before you begin

- Install IBM Mobile Test Workbench for Worklight and ensure that IBM Mobile Test Workbench for Worklight and Extension for Mobile are selected. For details, see Installing the product software.
- Install Xcode 4.6 or newer on a Macintosh computer running OS X v10.8 Mountain Lion or newer. Then, create an Xcode project that contains the source code of the application under test. Moreover, the `rtwBuildXcode.sh` script must be locally downloaded. It can be found in the build-script directory.

Procedure

1. In IBM Mobile Test Workbench for Worklight find the Workbench URL by clicking **File** > **New** > **Other** > **Test** > **Add Device**.

   You will need the Workbench URL to connect your Macintosh computer and your mobile device to the test workbench.

2. Connect your Macintosh computer to the test workbench, and then download the `RTW-iOS-Build-Archive.zip` build archive.

   The build archive contains scripts that are needed to prepare the app for testing.

   a. Open a browser window on your Macintosh computer and enter the Workbench URL in the following format:

      http://*Workbench_URL*:*port*/mobile

      For example, `http://192.0.2.24:7878/mobile`

   b. In the Rational Test Workbench - Mac OS Welcome page, click **Click here to start archive download**.

   c. When the download has completed, unpack the build archive.

      The following folders are included in the build archive: `browser, build-script, client, runtime, runtime-bundle`

3. Instrument the application under test by running the `rtwBuildXcode.sh script`.

   Instrumentation augments the AUT with code that allows you to record and play back a test.

   Note: The current user must have the appropriate permissions to run the script. If necessary, run the **`chmod`** command to change permissions.

   a. On your Macintosh computer, open the Macintosh Terminal application and change to the `build-script` folder.

   b. Type the following command to instrument the AUT and send it to the test workbench as an incoming application.

      ```
      <unpack_dir>/build-script/rtwBuildXcode.sh <.xcodeproj file> <Workbench URL>
      ```

      where

      - `unpack_dir` is the directory where you extracted the downloaded archive.
      - `.xcodeproj file` is the absolute or relative path to the .xcodeproj project created for the application under test. If the name or path to the Xcodeproj file contains spaces, enclose the full path with double quotation marks (" "), or prefix all spaces with backslashes (\).
      - `Workbench URL` indicates the Workbench URL copied from the Mobile Device editor. Including the Workbench URL is highly recommended, because this is the easiest way to register an application to the test workbench. The test workbench needs to know the application when it receives the recording log in order to produce a complete test. In a context in which the Workbench URL is used from the `rtwBuildXcode.sh` shell-script, do not include `mobile` at the end of the Workbench URL. As an example, indicate `http://<ip-address>:7878` only.

4. Add the instrumented version of the AUT to a project in the test workbench.

   Incoming applications must be managed to be used in a test. After you run the **`rtwBuildXcode.sh`** shell-script, do one of the following to manage the application:

   - Click the incoming application button in the test workbench.
   - In the Test Navigator, right-click an incoming application under Mobile Incoming Applications and click **Generate Managed Application**.

   Note: If you do not add the AUT as a managed app at this time, it will be done for you automatically when you generate a test at the end of the recording process for the AUT.

   For details, see “Importing applications to test in the workbench” on page 23.

5. Add the instrumented version of the application under test to your mobile device.
   a. On your mobile device, open the mobile test client in a browser by typing the full Workbench URL (`http://ip_address:port/mobile`), for example, `http://192.0.2.24:7878/mobile:7878/mobile`.
   b. From the mobile test client, tap **Manage Applications**.
   c. Select the application under test from the list.
   d. Click **Install**.

Results

After you complete these steps, you can start recording a test.

Instrumenting iOS applications on the iOS Simulator

Once the mobile test client is installed on the iOS Simulator, you must run a supplied build script on an OS Xsystem to be able to instrument the iOS application under test. The instrumented app can be optionally pushed to the test workbench and installed on the iOS Simulator.

Before you begin

You must have Xcode installed on your Macintosh computer running OS X v.10.8 (Mountain Lion) or newer, 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 IBM Mobile Test Workbench for Worklight click the **Display Workbench URL** icon and copy the Workbench URL.
2. On the Macintosh computer, open the web browser, paste in 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.
3. Open the Macintosh terminal application, 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.
4. 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 instrumented iOS applications on an iOS device

You can install an instrumented iOS application on a mobile device.

Before you begin

To push an instrumented iOS application to an iOS device, the application must first be added to the test workbench.

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.

   At the end of the installation, the instrumented application is installed on the device, ready for recording and testing. 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 **`-Xmxnnnnm`** parameter, where **`-Xmx`** is the command name, and `nnnn` 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 or simulator 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 mobile test client for Android.

You must have either added the Android application under test to the test workbench directly or uploaded the application from the mobile test client to the test workbench. For information about adding apps to the test workbench, see "Importing applications to test in the workbench" on page 23 for native or hybrid applications and "Adding web applications to test workbench" on page 25 for web applications. For information about uploading apps to the test workbench, see "Uploading Android applications from the mobile test client" on page 25.

About this task

This task applies to Android native applications, hybrid applications and web applications.

With the mobile test client for Android, you can record all actions on the user interface (UI), plus some actions on the phone itself: GPS locations, volume up and down, mute, headphones plug, and all media actions (play, pause, and so on), call, and end call. Camera and microphone functions are not supported.

Procedure

To record a session on the mobile device:

1. In the mobile test client, tap **Managed applications** for a native application or **Managed web applications** for a web application.
2. Tap an app in the list and tap **Record**.

   Note: If your device or emulator does not have silent mode, the mobile test client uninstalls the original version of the application under test and replaces it with the recording-ready app (instrumented version of your application). 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 and above (Android 4.2 +) 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**. Alternatively, click the link in the message that warns you that there is a new incoming recording. 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** and **Open test**. 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 34.

Related tasks:

"Getting started with Android testing" on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

Recording tests from the iOS mobile test client

Mobile tests are typically created by recording a session on the mobile device or simulator 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 the iOS Simulator if you do your testing in the iOS simulator using the native mobile test client. If you do your testing on a simulator or device using the browser-based client, you must type the Workbench URL in the Safari or Chrome browser on your device, or in the Safari browser on your simulator to run the client:

  `http://`*`Workbench_URL`*`:`*`port`*`/mobile`Carefully read the “iOS testing overview” on page 12 for more details.
- Native applications under test must be instrumented, and then pushed to the test workbench and simulator or installed on a device. For more details, see “Instrumenting iOS applications on the iOS Simulator” on page 29 and “Instrumenting iOS applications on an iOS device” on page 27 for each use case.
- If you are testing web applications, the IBM Rational Test Workbench Mobile Web Recorder is required to record user actions. You must download it from the Apple App Store on to your mobile device.

About this task

This task applies to testing iOS native applications, hybrid applications or web applications.

With the mobile test client for iOS, GPS hardware actions are supported.

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** to record a test for a native application, or tap **Manage web applications** to record a test for a web application.
2. Tap the application for which a test must be recorded. You might have to refresh the list to see the application. Then 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**. Alternatively, click the link in the message that warns you that there is a new incoming recording. 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** and **Open test**. The test editor opens in Rational Test Workbench and displays the generated test.

Related concepts:

"iOS testing overview" on page 12
With the mobile test client for iOS you can test native iOS applications, web applications, and hybrid applications. From your iOS device and from the iOS Simulator you can connect to the test workbench, record and run tests, and view reports.

Related tasks:

"Getting started with testing on the iOS Simulator" on page 20
Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

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 for the Android mobile client or Configuring the iOS mobile test client on the iOS Simulator 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 **Create a test from an existing recording** and select a recording session and select **Mobile Test** or click **Create a test from a new recording** 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 Applications** or **Managed web applications**.
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 and customize a mobile test imported in the test workbench or uploaded from a mobile device or simulator after recording.

The mobile test editor displays the test scripts. The edited test displays the list of actions and UI elements uploaded from a mobile device during the recording phase. Actions are represented in natural language, which allows you to modify 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 the screen captures of the user interface of the app during the recording. You can use the Mobile Data view to select user interface (UI) elements and add some verification points, or variables and modify steps in the test with simplified scripts. Or you can create or modify a set of steps manually directly in the test script.

You can add datapools, test variables, verification points, or stubs in your script.

CAUTION: The mobile client does not support data substitution. When a test is run from the mobile client, the values displayed are those entered during the recording, references and substitutions are not generated in the test. You must run the test from the workbench to verify substitutions.

Creating verification points in a test

You can create verification points for any 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. You can create verification points while recording a script or afterwards.

Before you begin

You can create verification points in the tests that are created from 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 41.

About this task

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 capture 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.

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 41.

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

You can use the test editor to add application stubs manually in your test.

About this task

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 application as a template for your application stub.

You can create stubs for Android, hybrid or iOS applications.

Example: To illustrate the use of stubs in a mobile application: When you tap 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 to 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.

Defining a variable in a test to run the latest version of an application

When you import or upload two versions of the same application to your workspace, by default the preferred version of the application to replay a test will be the earlier version, if this earlier version is used to record the test. This preference is specified in the launch application action of the test, and you can change the preference so that the latest version of the application available in your workspace is used to replay your test. To modify the behavior, define a variable with a mobile reserved name in your test and set the variable to a specific value. This ensures that, for a test launched in an automated test environment such as IBM Rational Quality Manager or from a command-line, the latest imported version of the application is selected without modifying the linked application in the test.

Before you begin

Open a test in the workbench from the Test Navigator view.

About this task

You must define a variable whose name is *RTW_Mobile_App_Selection* in your test and assign the `AlwaysUseLatestVersion` value to the test variable. If you do not have any container for your variables in your test, learn how to create one based on the procedure described in Defining a variable in a test to select a mobile device.

Procedure

1. To define the *RTW_Mobile_App_Selection* variable:
   a. In the test script, select the test variable node.
   b. Click **Insert** > **Variable Declaration**.
   c. Enter the name of the variable *RTW_Mobile_App_Selection* and click **OK**. The variable is added to the test variable container.
   d. In the **Visible in** section, select **This test only** to restrict data to the current test. Optionally 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.
2. Assign the `AlwaysUseLatestVersion` value to the variable:
   a. Click **Initialize value to** and enter the value `AlwaysUseLatestVersion` in the **Text** field. This will select the latest imported application version, not the one the test is linked to or defined as the preferred application.

   b. Save the test.

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 41.

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.

The other way is to assign a variable to an object select in the Data Mobile view but the variable must be created before:

5. In the Mobile Data view, from the **Screen Capture** tab or the **Elements** tab, right-click an object and select **Create variable assignment from** the element selected. In the wizard that opens, select a variable and click **OK**. The variable is added to the test suite.

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. See 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

After generating a test from a recording, you can run the test on multiple mobile devices and simulators on which the mobile test client is installed. Tests created for native or hybrid applications must be run on the Android mobile test client if recorded on Android platform, and on the iOS mobile test client if recorded on iOS platform. But there is no such restriction for tests of IBM Worklight hybrid applications, as they can be recorded on Android and played back on iOS platform, or recorded on iOS and played back on Android mobile test client. If you test web applications, the IBM Rational Test Workbench Mobile Web Recorder is used to record and run your tests on Android and iOS mobile test client. Note that all tests can be run from the test workbench.

Before you begin

CAUTION:
The mobile client does not support data substitution. When a test is run from the mobile client, the values displayed are those entered during the recording, references and substitutions are not generated in the test. You must run the test from the workbench to verify data substitution.

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.

About this task

This task applies to native, hybrid and web applications recorded on an Android mobile test client that you run on a simulator or mobile device. You can play back tests for web applications on the Android or iOS platform.

CAUTION:
You can record a test on a device such as a phone and play back the test on another type of device (for example, on a tablet) only if the application tested has the same behavior on both types of devices.

With the mobile test client for Android, you can play back GPS locations and hardware actions such as the use of volume up and down, mute, the use of headphones, and all media actions (play, pause, and so on), making calls, and ending calls. Camera and microphone functions are not supported.

Procedure

1. In the mobile test client, tap **Managed Applications** for a native application or **Managed web applications** for a web application, and then 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**. If your device or emulator does not have silent mode, the recording-ready app is uninstalled and replaced with a playback-ready app. 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. Silent mode is not available on devices and emulators with API level 17 and above (Android 4.2 +) due to a known limitation.

You can simplify this process of installing and uninstalling versions of the Android app by choosing **Playback on instrumented** from the Settings page on your Android device or emulator. This lets you play back a test using the more heavily instrumented recording version of the app, rather than the lighter weight playback version of the app. This is at the expense, however, of slower playback speed and greater memory consumption. The test is played back in the mobile device. Do not interact with the mobile device until the test is complete.

What to do next

You can now evaluate the test results. See Evaluate results.

Related concepts:

"Android testing overview" on page 11
With the mobile test client for Android you can test native Android applications, web applications, and hybrid applications from your Android device and from Android emulators.

Related tasks:

"Getting started with Android testing" on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

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 .
- 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 the iOS Simulator if you do your testing in the iOS simulator using the native mobile test client. If you do your testing on a simulator or device using the browser-based client, you must type the Workbench URL in the Safari or Chrome browser on your device, or in the Safari browser on your simulator to run the client:

  `http://Workbench_URL:port/mobile`Read carefully the "iOS testing overview" on page 12 for more details.
- IBM Rational Test Workbench Mobile Web Recorder must be installed on the device or simulator prior to recording web applications.

About this task

This task applies to testing iOS native applications, hybrid applications or web applications.

With the mobile test client for iOS, you can play back all actions on the user interface and GPS locations.

Restriction: You can record a test on a device, for example on a iPhone and run the test on another type of device, for example on an iPad, only if the application under test is the same for both type of devices.

Procedure

To run a test from the iOS device or simulator:

1. On your device or simulator, tap **Manage Applications** to test a native or hybrid application or tap **Manage web applications** for testing a web application. Then, tap the application under test.
2. You can see the list of tests available for the app that shows up, tap the test name.
3. The device or simulator displays details on the recording and the test suite. Tap **Run Test**. The test is run on the device or simulator. Do not interact with the simulator or device until the test is complete.
4. When playback is complete, the mobile test client is launched.

What to do next

You can now evaluate the test results. For more information, see Evaluate results.

Related tasks:

"Getting started with testing on the iOS Simulator" on page 20
Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

Running tests from the test workbench

After generating the test from a recording, you can edit a test according to your requirements and play it back from the workbench on your mobile device/simulator. This means that the playback is controlled from the workbench and not from the simulator or mobile device. The result of the tests can be viewed from both the test workbench and the mobile test client.

Before you begin

- You must have recorded and generated a test as mentioned in "Recording tests from the test workbench" on page 34.
- Ensure that you give control to the test workbench to run tests by tapping **Enter passive mode**. After you tap it, on the mobile client, a message tells you to press **Back** to end this mode or wait until a playback scenario starts. On iPads, you must select any other page on the left to end the passive mode.

Note: If your test contains datapools, it will be controlled by the workbench.

About this task

This procedure applies to Android and iOS applications. You can play back a test of web application on either platform. You can also run the same test recorded for an Android or iOS IBM Worklight hybrid application and run it from the test workbench on either platform.

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. A wizard displays the list of available devices. Select the device on which the test will be run.
3. 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.

   Note:

   Do not interact with the mobile device till the test is complete.

   If you need to stop the test run, click the **Stop test** button in the toolbar of the test workbench.

What to do next

You can now evaluate the test results. See Evaluate results.

Related tasks:

"Getting started with Android testing" on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

"Getting started with testing on iOS devices" on page 17
You can test native, hybrid, and web applications on an iOS device using the browser-based client for iOS. This client is a web application that runs in the Safari or Chrome browser on your device.

"Getting started with testing on the iOS Simulator" on page 20
Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

Running Worklight hybrid tests on either Android or iOS mobile test client

To test Worklight hybrid applications, you can record a test on a mobile device or simulator where the native Android or iOS mobile test client is installed and run the same test on either platform.

Before you begin

You must import the application that you want to test into the test workbench for both platforms Android and iOS. The same version is required. You must record and generate a test. For more details, see "Recording tests from the Android mobile test client" on page 31 or "Recording tests from the iOS mobile test client" on page 32.

Procedure

To run the test:

1. Tap **Manage Applications** and tap the application under test.
2. Tap refresh so that the test shows up in the list of tests.
3. Open the test. The device displays details on the test. Tap **Run Test**. Do not interact with device until the test is complete.

What to do next

When the test is complete, you can see the report on the device. For more information, see Evaluate results for details. Buttons, labels, button clicks, and their associated text properties are supported on both platforms. When a native step cannot be translated and run on the other platform, as is the case for the Android back button, for example, a yellow warning advises you in the report that this Android step cannot be translated and run on iOS because this may cause a test failure.

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

This task applies to Android and iOS applications under test.

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 the correct locale for the translation of the strings. This must be the local string used on the device during the test recording.

    Note: You can have device specific Locales in the list, for iOS devices. For example, if you recorded a test on a device set to English Locale, there could be other choices of locales other than just En for English: en_iPhone (this should be selected if recording was done on iPhone) and en_iPad (this should be selected if the recording was done on iPad).
    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 local string 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 tasks:

"Recording tests from the Android mobile test client" on page 31
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

"Recording tests from the iOS mobile test client" on page 32
Mobile tests are typically created by recording a session on the mobile device or simulator 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.

"Recording tests from the test workbench" on page 34
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.

"Running tests from an Android mobile test client" on page 44
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.

"Running tests from the iOS mobile test client" on page 45
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.

"Running tests from the test workbench" on page 46
After generating the test from a recording, you can edit a test according to your requirements and play it back from the workbench on your mobile device/simulator. This means that the playback is controlled from the workbench and not from the simulator or mobile device. The result of the tests can be viewed from both the test workbench and the mobile test client.

"Viewing mobile reports" on page 51
You can choose to view the test reports on the mobile device, on an emulator, or on the test workbench.

"Getting started with Android testing" on page 15
Use this topic to help you get started with your testing of applications that run on Android devices.

"Getting started with testing on iOS devices" on page 17
You can test native, hybrid, and web applications on an iOS device using the browser-based client for iOS. This client is a web application that runs in the Safari or Chrome browser on your device.

"Getting started with testing on the iOS Simulator" on page 20
Use this topic to help you get started with your testing of native, hybrid, and web applications on the iOS Simulator.

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, at the end of play back, the report opens up automatically on 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, for native applications, hybrid applications and web applications under test.

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 **Manage Applications** for a test from a native app and **Manage Web Applications** for a test from a web app.
   b. Select the app for which you want to view the results.
   c. Tap a test suite name and then tap **Reports** tab button.
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. Depending on how you executed a test, the results are stored in the appropriate folder. When running the test from the mobile device or emulator, the results are in the **Mobile Results** folder. When tests are run from the workbench, the results are in the **Results** folder. Note that these are two separate folders within the logical view of the Test Navigator. Click to open the logical view.

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 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 test workbench project.
2. In the Test Workbench perspective, in the Test Navigator, right-click the test workbench 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 test workbench 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 test workbench project, visible in the Logical View. The new test is also visible in the Resource View, under the test workbench 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 int 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 test workbench project. If you have an existing compound test, you can import the test to a test workbench project.

Creating a compound test in a test workbench project

You can create a compound test in a test workbench project.

Procedure

1. Create a test workbench project.
2. In the Test Workbench perspective, in the Test Navigator, right-click the test workbench 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 test workbench 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 test workbench project, visible in the Logical View. The new test is also visible in the Resource View, under the test workbench 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 test workbench project.

Procedure

1. In the Test Workbench perspective, in the Test Navigator, right-click the test workbench 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 test workbench 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.

Adding compound tests to schedule

This topic is applicable for IBM Rational Performance Tester version 8.5.1. To test the performance of multiple tests, you can add all the tests to a compound test and add the compound test to a user group. When you run a schedule, all the tests in the compound test are run in a sequential order.

About this task

A user group can be defined locally, which means the parameters to run the user group are defined in the schedule. A user group can be associated with a compound test, which means that the parameters to run that user group are defined in the compound test.

Procedure

To add a compound test:

1. In the schedule editor, add a user group.
2. Click the user group and in **Behavior**, click **Use compound test**.
3. Select a compound test and click **OK**. If there are no compound tests in the project, click **Create**, specify a name for the compound test, and click **Finish**. If there are test variables associated with a compound test and also defined in the schedule, the variables with the compound test take precedence while running the user group.

4. Save the test.

Related concepts:

Schedule overview

Extending Rational Test Workbench Eclipse Client

You can extend the capabilities of test workbench by creating custom Java code.

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 `VerifyYUserID.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 test workbench. 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 test workbench. 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.
ITestExecutionServices	Provides information for adding custom test execution features to tests. Replaces the IKLog interface. All the methods that were available in IKLog 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.

Test execution services interfaces

Interface	Description
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.
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.

Test execution services interfaces

Interface	Description
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`.
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.

Test execution services classes

Class	Description
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(IVirtualUse
 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, seeHow loops affect the state of virtual users.

```
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 varia
        tes.getValue("myLocalVar", tes.STORAGE_LOCAL);  // This variable is stored locally, per test

        tes.setValue("myVar", tes.STORAGE_USER, "myNewValue");  // Change the value of the variable r
        tes.setValue("myLocalVar", tes.STORAGE_LOCAL, "myLocalNewVar");  // Change the value of the 
        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 the
  * Help menu and select 'Extending Rational Performance Tester functionality' -> 'Extending test exe
  */
 public String exec(ITestExecutionServices tes, String[] args) {tes.getStatisticsManager().getStatTre
  .submitDataPoint(Double.valueOf(Math.random()*100.).longValue());

  return null;
 }

}
```

Related tasks:

"Creating custom Java code" on page 60
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.

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("Invester 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 complier 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, `Schtest`.
    a. In `Schtest`, 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
-Xrunjdwp: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

Chapter 7. 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 sometimes upgrading your apps. Most of this upgrade is automatically done for you when you use the installer. However, the upgrade might also involve some manual operations, such as setting various properties, using special command facilities, and running supplied Ant tasks. The complete upgrade procedures are detailed in the following topics.

Migrating from IBM Worklight V5.0.6 or later to V6.1.0

This section contains the procedures for upgrading from IBM Worklight V5.0.6 or later to V6.1.0 and migrating your Worklight applications to work with IBM Worklight V6.1.0.

The topics in this section of the documentation focus on the migration process from IBM Worklight V5.0.6 or later to IBM Worklight V6.1.0.

They cover how to upgrade your IBM Worklight Studio and IBM Worklight Server software to V6.1.0, and how to migrate your applications for V6.1.0.

- For information about how to upgrade your current installation of Worklight Studio to V6.1.0, see “Upgrading to Worklight Studio V6.1.0” on page 222.
- For information about how to upgrade your current installation of Worklight Server to V6.1.0, see “Upgrading to Worklight Server V6.1.0 in a production environment” on page 226.

If you are upgrading from versions of IBM Worklight earlier than V5.0.6, or to an IBM Worklight version other than V6.1.0, see the following sections, as appropriate:

- “Migrating from IBM Worklight V5.0.6 to V6.0.0” on page 259
- “Migrating from IBM Worklight V5.0.5 to V5.0.6” on page 303
- “Migrating from IBM Worklight V5.0.0.3 to V5.0.5” on page 308

Separation of lifecycle between Worklight Server and Worklight Studio

Since IBM Worklight V6.1.0, there is a separation of the Worklight Studio and the Worklight Server upgrade lifecycles, which provides benefits to both developers and IT staff.

In version 6.1.0, IBM Worklight allows a separation between the Worklight Server and Worklight Studio lifecycles. This separation means:

- It is possible to upgrade an instance of Worklight Server to version 6.1.0 without upgrading your existing applications to Worklight Studio version 6.1.0.

- It is possible to deploy project WAR files, apps, and adapters that are developed with any supported version of Worklight Studio (V5.0.6.x, V6.0.0.x, and V6.1.0) to an instance of Worklight Server V6.1.0. But the apps that are deployed must be built with the same version of Worklight Studio as the project WAR file that was previously deployed. As an example, consider a Worklight project that was developed on Worklight Studio V5.0.6, and this project's WAR file is later deployed on Worklight Server V6.1.0. This WAR file is deployed as a V5.0.6 project, without having been opened in Worklight Studio V6.1.0. You can later deploy to this Worklight project only applications and adapters that were developed using Worklight Studio V5.0.6 (and not V6.1.0 or any other version).

Some limitations of this lifecycle separation are as follows:

- Only application environments that are supported by Worklight Server V6.1.0 can be migrated. Older application environments that are not supported by Worklight Server V6.1.0 (for example, iGoogle, Windows Phone 7.5, or Facebook) will no longer be available after the server upgrade.
- To deploy a project WAR file, you must use the tools that are provided with the target Worklight Server version you are deploying to. That is, to deploy with an Ant task to Worklight Server, you must use the `worklight-ant-deployer.jar` file that is located in the `WorklightServer` directory of the Worklight Server installation directory.

Terminology

In the topics that deal with migration and updating, the following definitions of several important terms are used:

- *Upgrade* – Moving from one version of software to the next. For example, you upgrade an installation of Worklight Server V6.0.0 to Worklight Server V6.1.0.
- *Migrate* – Updating, either automatically or manually, a piece of software so that it is able to use the next level of the software. For example, you migrate a Worklight project's database schema to use the next version level of Worklight Server. Or you migrate a Worklight application to use the next version level of Worklight Studio.
- *Deploy* – Installing an application on a server. For example, you *deploy* a Worklight application to a production instance of Worklight Server running on an application server.

Upgrade paths

The topics under this section apply to the following types of upgrade and migration paths:

- *Major version change* – For example, upgrading from V5.0.6.x to V6.1.0.
- *Minor version change* – For example, upgrading from V6.0.0 to V6.1.0.
- *Fix pack upgrades* – For example, upgrading from V6.1.0 to V6.1.0.x.
- *Interim fix* – For example, upgrading from V6.1.0 to an interim fix identified by a build number.

Upgrading to Worklight Studio V6.1.0

How to upgrade your current version of Worklight Studio to the latest version.

About this task

The upgrade to Worklight Studio V6.1.0 is performed as an Eclipse P2 update operation. After Worklight Studio V6.1.0 is installed, you can then point to your earlier workspace and work with your existing projects.

Procedure

1. Start your Eclipse IDE workbench and verify your version of Eclipse.
   - Worklight Studio V6.1.0 must be installed into Eclipse V4.2.2 (Juno) or Eclipse V4.3.1 (Kepler) as a new installation. It cannot be installed in earlier versions of Eclipse (prior to Juno).
   - If you have an older version of Eclipse, update it to Juno or Kepler before continuing this procedure.
2. Click **Help** > **Install new software**.
3. In the Add Repository window, click **Archive**.
4. Browse to the update site directory on the installation disk or to your downloaded installation files.
5. Select the update site `.zip` file and then click **OK**.
6. On the Available Software page, select **IBM Worklight Studio Development Tools**, and click **Next**. If you want to see the components to be installed, expand **IBM Worklight Studio Development Tools**, and select the components you want:
   - Always select **IBM Worklight Studio**.
   - Select **IBM Dojo Mobile Tools** if you anticipate using that JavaScript library.
   - Select **IBM jQuery Mobile Tools** if you anticipate using that JavaScript library.
7. On the Install Details page, review the features of Worklight Studio to be installed.
   a. You may see one or more messages in the lower part of the page similar to `Your original request has been modified. "IBM Dojo Mobile Tools" is already installed, so an update will be performed instead.` This is expected, and indicates that an update is being performed.
   b. Click **Next**.
8. On the Review Licenses page, review the license text. If you agree to the terms, select **I accept the terms of the license agreement** and then click **Finish**.
9. The installation process starts. Follow the prompts (during which you may be asked to restart Eclipse) to complete the installation.

Results

Worklight Studio is now updated.

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

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

The following topic contains information about how to migrate and work with your existing projects.

Migrating Worklight projects to Worklight Studio V6.1.0

How to migrate your existing projects (including those created in Worklight Studio V5.0.6.x or V6.0.0.x) to Worklight Studio V6.1.0.

Migrating older Worklight projects to Worklight Studio V6.1.0

Open your existing projects (that is, projects originally created in Worklight Studio V5.0.5, V5.0.6.x or V6.0.0.x) as you would normally. This action triggers a migration process that updates them to work with V6.1.0.

Note: To migrate projects that were created in versions of Worklight Studio older than V5.0.5, you must first migrate these projects to an intermediate level such as V5.0.5, V5.0.6, or V6.0. For example, if you have a V5.0.0.3 project:

1. Migrate the V5.0.0.3 project to Worklight Studio V5.0.5.x, using the procedures listed in "Migrating from IBM Worklight V5.0.0.3 to V5.0.5" on page 308.
2. Open the project (now migrated to V5.0.5) in Worklight Studio V6.1.0 to complete the migration process.

If for any reason you need to access the pre-migrated versions of your Worklight projects, a compressed file backup is made of those files. The location of this file is displayed in the second step of the migration procedure.

If any of your existing target environments are removed in the newest version of Worklight Studio, a message notifies you, and those folders are marked as plain source folders in your Worklight file hierarchy.

If any applications in your existing projects use the obsolete database login module for user authentication, modify them to use adapter-based authentication with the SQL adapter instead.

Important: In IBM Worklight V6.1.0, the following elements of the JavaScript client-side API, which were deprecated in previous versions of IBM Worklight, are now removed:

- `WL.Page`: Removed
- `WL.Fragment`: Removed

As a replacement to build your multi-page applications, consider using the equivalent implementation in JavaScript frameworks such as jQuery Mobile, Sencha Touch, and Dojo Mobile. You might also use embedded jQuery APIs to load page fragments (see http://api.jquery.com/load/). For more information about how to build a multi-page application, see the module *Building a multi-page application* under category 3, *Worklight client-side development basics*, in Chapter 3, "Tutorials and samples," on page 27.

Migrating Worklight Native API projects

Projects of type **Worklight Native API** that were created in earlier versions of Worklight Studio have the JAR files containing their native code rebuilt when they are migrated to Worklight Studio V6.1.0. As a result, after you migrate your existing Native API project, you must recopy the library and the client property files of your Native API application into your Worklight project.

- For iOS Native API projects, follow the instructions in "Copying files of Native API applications for iOS" on page 471.
- For Android Native API projects, follow the instructions in "Copying files of Native API applications for Android" on page 474.

- For JavaME Native API projects, follow the instructions in “Copying files of Native API applications for Java Platform, Micro Edition (Java ME)” on page 478.

Working with Android projects

In IBM Worklight V6.1.0, developers are encouraged to use the latest Android SDK API level that is supported by the Worklight Studio – Android 4.3 (API Level 18). Using the latest Android SDK allows the Android system to disable compatibility behaviors that slow the mobile application and to use the latest capabilities and features it includes.

When a new Android project is created, an attribute named `android:targetSdkVersion` is added in the `androidManifest.xml` file under the `<uses-sdk>` element, with a default value of 18. This value specifies that the API Level of the application targets is Android 4.3. In previous releases of Worklight Studio, this value was not specified, and the API level 8 was targeted by default.

The default Android SDK API level is not changed for existing projects that are opened in Worklight Studio V6.1.0.

Note: The Cordova libraries are updated during the installation of Worklight Studio V6.1.0. Therefore, for Android applications, if you have any user/custom plug-in that refers to the `org.apache.cordova.api` package, you must replace `org.apache.cordova.api` with `org.apache.cordova`.

Impact of migrating to a new version of Worklight Studio for applications already in production

In order for Direct Update to work, both the client application and the server-side artifacts must be generated from the same version of Worklight Studio.

In the following cases, if you migrate your Worklight project to a new version of Worklight Studio, and even if you do not change the code of the application, you must still increment the version number of the application. After you make your new application available to the Worklight production server, you must ask your users to download a new version of the application from the application store.

- Applications that were created with a version of Worklight Studio older than V5.0.0.3. The communication protocol of Worklight Server V6.1.0 supports the protocols of client applications that are built with IBM Worklight V5.0.0.3 or later. Device users who use apps that were built with IBM Worklight V5.0.0.3 or later, and whose server-side artifacts have been successfully deployed to Worklight V6.1.0 and tested on a test server should continue to work without requiring the device user to download a new version of the application.

  Device users who use applications that were built with IBM Worklight V5.0.0.3 or later, whose server-side artifacts were regenerated using the newer version of Worklight Studio, and that are successfully deployed to Worklight Server V6.1.0 and tested on a test server should continue to work without requiring the device user to download a new version of the application. However, Direct Update is not available for these applications.
- Applications that use the Direct Update feature. The Direct Update feature (“Direct updates of app versions to mobile devices” on page 833) to automatically update application versions stops working after some migration paths. Worklight Studio V6.1.0 displays a warning when such situations are detected when migrating apps created in older versions.

To notify your users that a new version of the application is available, you can use the startup display notification feature that is documented at “Displaying a notification message on application startup” on page 840. If the application update is mandatory, another alternative is to deny access to the old application version with the feature that is documented at “Remotely disabling application connectivity” on page 837.

If you need to upload a new version of your application to a public application store such as the Apple Store or Google Play, you must in some cases resubmit the app for approval by the store.

Upgrading to Worklight Server V6.1.0 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. This section provides a series of steps to upgrade your production server or servers efficiently and in the shortest time possible.

When you upgrade from Worklight Server V5.0.6.x or V6.0.0.x to V6.1.0 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 the topics that follow 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 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 backing up any existing databases or application server data, migrating your existing IBM Worklight projects and applications to the new version, 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 IBM Worklight applications.

In the following topics, you can also find specific instructions for upgrading from Worklight Server V6.1.0 to V6.1.0.1 (fix pack), or to an interim fix.

Read through 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.

Start with “Overview of the upgrade to Worklight Server V6.1.0 process” on page 227, and then read through the steps under each of the major topics that follow.

Overview of the upgrade to Worklight Server V6.1.0 process

An overview of the Worklight Server V6.1.0 upgrade process, including what is updated and what is not.

A typical instance of Worklight Server includes the following elements:

- A Database Management System (DBMS) that runs databases for the IBM Worklight Application Center and for Worklight Server.
- One or more application servers. These application servers host and run:
  - The IBM Worklight Application Center application (if Application Center is installed on that server).
  - One or more Worklight Server applications. Each Worklight Server application:
    - Is defined by a WAR file that contains a Worklight Console, default values for server-specific configuration settings, and other resources that are required to run the Worklight applications and adapters.
    - Is connected to two databases, one for Worklight and one for Worklight Reports.
    - Can run on one or more physical servers (for both workload and service availability considerations).
  - An installation of the IBM Worklight Server programs, usually on the same computer as the Application Server.

Other items can belong to a Worklight configuration, for example, an IBM HTTP Server, IBM DataPower, or an LDAP system.

But the topics in this section are only focused on the task of upgrading and configuring the following entities:

- The IBM Worklight Server programs.
- The databases.
- The Worklight Server and Application Center applications that are deployed in the Application Server.

The actual steps that you must complete for the upgrade can change, depending on the particular *upgrade path* you are pursuing. Your upgrade path is determined by whether you are upgrading:

- From a previous version of IBM Worklight to Worklight Server V6.1.0 (for example, from V5.0.6.x to V6.1.0 or from V6.0.0.x to V6.1.0).
- From Worklight Server V6.1.0 to a fix pack release (for example, from V6.1.0 to V6.1.0.x).
- From Worklight Server V6.1.0 to an interim fix release (for example, from V6.1.0 to a designated interim fix).

The spreadsheet at the following link lists the individual steps for each of these upgrade paths, and helps you to determine:

- Whether the step is required or not required, depending on your IBM Worklight upgrade path.
- Whether your Application Center and Worklight Server is running (old version), uninstalled, stopped, or upgraded (and running) during this step as the result of actions in the current step or previous steps.

The spreadsheet can be downloaded here: Worklight_Server_V610_Upgrade_Steps spreadsheet

To provide further assistance, at the beginning of each topic a shorter version of this spreadsheet is provided for that step.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes/No	Yes/No	Yes/No	Yes/No	Running/ Stopped/ Uninstalled/ Upgraded	Running/ Stopped/ Uninstalled/ Upgraded

Preparing for the upgrade to Worklight Server 6.1.0

Several preparation tasks must be completed before you begin the actual installation of Worklight Server V6.1.0.

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 are required for the upgrade, to minimize production downtime and inconvenience to users of existing applications.

The following topics cover preparation tasks to be completed before you begin the installation of the new Worklight Server version. These tasks can be performed at any time before the upgrade, but must be completed before you move to the next major step, “Starting the Worklight Server V6.1.0 upgrade process” on page 238.

Gathering the information you need for the Worklight Server V6.1.0 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.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Running	Running

Procedure

Go through the following checklist to make sure that you have all of the necessary information to begin the upgrade:

- Verify that your operating system, application server, and database meet the system requirements for Worklight Server V6.1.0 at Detailed System Requirements for IBM Worklight and IBM Mobile Foundation.
- 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 for the target databases are available on your computer. Access to them is needed for IBM Installation Manager to redeploy the Application Center application, and for the Ant scripts to redeploy the Worklight Console.
- 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.
- Identify all systems that you must update if the URL to access the Application Center or the Worklight Server application or their console changes. If you upgrade from V 5.0.6 to V 6.1,0, the URL changes. It can also change if you plan to perform a rolling upgrade.

Identify the Worklight WAR file and prepare the Ant deployment script:

In this task, you identify the IBM Worklight WAR file that contains numerous resources and configuration settings for Worklight Server, and prepare the Ant script that is used to deploy it.

About this task

The Worklight WAR file is a web application archive that contains a Worklight Console, default values for server-specific configuration settings, and other resources that can be required to run the Worklight applications and adapters.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Running	Running

Procedure

In the upgrade process, the Worklight project WAR file must be redeployed to the application server. It is important to deploy the same WAR file. To ensure this, you must complete the following steps:

- Find the WAR file that was previously deployed to the application server.
- If you are upgrading from Worklight Server V6.0.0.x, find the JNDI properties that were set for the deployed Worklight project to override the default `worklight.properties`. If you used an Ant script with the task `configureapplicationserver` to deploy the WAR file, you can find the JNDI properties that were set at installation time in this script. For more information, see "Configuration of IBM Worklight applications on the server" on page 772.

For upgrading from Worklight Server V6.0.0.x, the procedure to deploy the WAR file is defined at "Deploying the project WAR file" on page 714

When you upgrade from Worklight Server V5.0.6.x, a Worklight WAR file is installed by the installer. If you have not modified this WAR file on your production server, you must create a modified file by following the instructions at "Building a project WAR file with Ant" on page 714.

- When you modifying the WAR file, use Worklight Studio V5.0.6.x or the Ant tasks (`worklight-ant.jar`) from an installation of Worklight Studio V 5.0.6.x that was used to build the apps previously deployed to the server. The version of Worklight Studio used to build the project WAR file must match exactly the version of Worklight Studio used to build the apps previously deployed to the server.
- The WAR file is automatically migrated to V6.1.0 format during the deployment procedure that is defined in later steps.

You must also prepare the Ant deployment script that is used to migrate this war file to version 6.1.0, and to deploy it to the application server, with the upgraded Worklight Runtime Library.

When you upgrade from Worklight Server V6.0.0.x, you can reuse the script that you used for the initial deployment. Make a copy of this file and apply to it the following modifications:

- In the Ant file, create a target named `minimal-update`, with the following content:

```
<target name="minimal-update">
  <updateapplicationserver>
     -----> (Copy here the same content as in task <configureapplicationserver>)
  </updateapplicationserver>
</target>
```

  The minimal-update target can be used in some upgrade paths. When this target can be used, the configuration of the application server is not changed and it is not necessary to review the configurations of the deployed application or datasources. See Worklight_Server_V610_Upgrade_Steps spreadsheet to determine the upgrade paths in which this scenario can be used.

- In the Ant file, replace the reference to the JAR file that defines the IBM Worklight tasks. In Worklight Server V6.0.0, the JAR file is named worklight-ant.jar. In Worklight V6.1.0, it is named worklight-ant-deployer.jar. For example, if the V6.0.0 script looks like this:

  ```
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <fileset dir="${worklight.server.install.dir}/WorklightServer">
        <include name="worklight-ant.jar"/>
      </fileset>
    </classpath>
  </taskdef>
  ```

  replace the reference to the JAR file, as highlighted below:

  ```
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <fileset dir="${worklight.server.install.dir}/WorklightServer">
        <include name="worklight-ant-deployer.jar"/>
      </fileset>
    </classpath>
  </taskdef>
  ```

When you upgrade from Worklight Server V5.0.6.x:

- Install Worklight Server V6.1.0 on your computer, but *without installing Application Center*.
- Navigate to directory <*WorklightInstallDir*>/WorklightServer/configuration-samples.
- Select the file that corresponds to your combination of application server and database (the files are named with the convention redeploy506-<*appserver*>-<*db*>.xml).
- Make a copy of this file.
- Edit the copied file and change the values of the properties to match your installation configuration.

Alternatively, you can create an Ant file using the Server Configuration Tool and export the resulting Ant file to run it on the command line.

The Server Configuration Tool does not separate calls to the individual Ant targets as this is required in the upgrade procedure. If you create an Ant file with the Server Configuration Tool, review the database settings carefully. If you enter different database settings than those that were used at install time, the deployed apps will not be visible to the Worklight Server.

Review and note the Application Server configuration for Worklight Server and Application Center:

In this task, if it is required for your upgrade path, you prepare for the undeployment and redeployment of applications to the application server to correct information that can potentially be modified or deleted by IBM Installation Manager.

About this task

In some upgrade scenarios, the applications that are deployed to the application server must be undeployed, and then redeployed. As a consequence, the configurations that were previously made to these applications are erased and must be reconfigured after the application is deployed again to the application server.

The applications to review are as follows:

- For Application Center:
  - The Application Center database
  - The Application Center Console and Application Center Services
- For Worklight:
  - The Worklight Console (or, in WebSphere Application Server Full Profile, `Worklight Console <id>` if you specified an id at deployment time)

The JDBC data sources to review are as follows:

- For Application Center: the Application Center database
- For Worklight:
  - The Worklight Database (or, in WebSphere Application Server Full Profile, `Worklight database <id>` if you specified an id at deployment time)
  - The Worklight Reports Database (or, in WebSphere Application Server Full Profile, `Worklight reports database <id>` if you specified an id at deployment time)

If these items were previously configured, note the configuration details so you can reconfigure them after the applications are reinstalled and redeployed. The configurations affected can include security settings, lists of users authorized to use the application, startup behaviors, connection pool settings, JNDI properties, and other items.

The upgrade paths in which this step is not mandatory are listed in the following table.

Is this step required for your upgrade path?				**System status after this step, if both Application Center and Worklight Server are on the same application server**	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	No (for Worklight Server), Yes (for Application Center)	See fix pack installation instructions	See interim fix installation instructions	Running	Running

Procedure

To review the configuration of the data sources and applications:

- For WebSphere Application Server Full Profile, use the console.
- For WebSphere Application Server Liberty Profile:

  - Open the server.xml file. The settings that can be modified or removed by IBM Installation Manager are between the marker comments, as shown in the following sample:

    ```
    <!-- Begin of configuration added by IBM Worklight installer. -->
    ...
    <!-- End of configuration added by IBM Worklight installer. -->
    ```

- For Apache Tomcat:
  - Open the server.xml and the tomcat-users.xml files. The settings that can be modified or removed by IBM Installation Manager are between the marker comments, as shown in the following sample:

    ```
    <!-- Begin of Context and Realm configuration added by IBM Worklight installer. -->
    ...
    <!-- End of Context and Realm configuration added by IBM Worklight installer. -->
    ```

Verify environments of deployed apps:

Before you upgrade to Worklight Server V6.1.0, verify that all of the environments that are targeted in your Worklight applications are still supported.

About this task

After the migration is completed, your Worklight applications will contain only the environments that are supported by the current version of Worklight Server.

For example, IBM Worklight Server V6.1.0 no longer supports some of the Worklight environments such as iGoogle, Facebook, Apple OS X Dashboard, Vista that were supported in IBM Worklight V5.0.6. If a target mobile device has an application that is installed on it which requires an environment that is no longer supported by Worklight Server V6.1.0, the application on this device will stop working after an upgrade of Worklight Server to version V6.1.0.

However, if your Worklight Server contains applications that have support for a mobile OS version that was released after the release of Worklight Server V6.1.0 (for example, because your development team applies a fix pack to Worklight Studio that provides support for this new mobile OS), these applications will work after the server is upgraded to V6.1.0. However, make sure that you deploy a Project's WAR file built with the exact same version of Worklight Studio as the version used to build the apps.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Running	Running

Procedure

If your current version of Worklight Server includes existing applications that target environments that are no longer supported by Worklight Server V6.1.0:

- For old, no-longer-supported environments, your application developers must update the Worklight application to run with an environment supported by Worklight Server V6.1.0 before you can run it.

- For new environments for which support is added after the release of Worklight Server V6.1.0, check for the availability of a fix pack release that provides support for this environment.

The following table can help to determine the IBM Worklight versions in which support for older environments was discontinued, and to suggest possible replacement environments for those environments.

Environment	Support removed in	Suggested replacement path
Facebook	IBM Worklight V6.0	Desktop web app
iGoogle	IBM Worklight V6.0	Review environments supported by IBM Worklight V6.1.0
Apple OS X Dashboard	IBM Worklight V6.0	Review environments supported by IBM Worklight V6.1.0
Windows 7 and Vista	IBM Worklight V6.0	Review environments supported by IBM Worklight V6.1.0
Windows Phone 7.5	IBM Worklight V6.1.0	Review environments supported by IBM Worklight V6.1.0

Deciding between in-place upgrade to Worklight Server V6.1.0 and rolling upgrade:

There are two ways to perform an upgrade: *in-place upgrade* or *rolling upgrade*.

About this task

- An in-place upgrade is an upgrade by which the old version of Worklight Server is no longer installed after the new version of Worklight Server 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 Worklight Server in the same application server or in a different application server.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Running	Running

Procedure

The in-place upgrade is the normal way to upgrade. Its advantage is 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 with the old configuration and a backup of the databases while investigating the problem.

Important: This release supports the rolling upgrade for Worklight Server and Worklight projects. However, a rolling upgrade of Application Center is not supported.

This chapter focuses on the in-place upgrade. It mentions some specific instructions for rolling upgrade, but the rolling upgrade is complex and is not yet fully explained in this documentation.

Packaging change of WebSphere Application Server Liberty Profile in IBM Worklight V6.x:

Important information about how WebSphere Application Server Liberty Profile is delivered since IBM Worklight V6.0.0, and how it can impact the upgrade of your production Worklight Server.

About this task

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 V6.1.0, WebSphere Application Server Liberty Core is not embedded in the IBM Installation Manager wizard of Worklight Server. Instead, it is provided as a separate IBM Installation Manager wizard.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes	No	No	No	Running	Running

Procedure

As a result of this packaging, the Worklight Server upgrade process 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 IBM Installation Manager wizard of Worklight Server.

To prevent this existing server from being uninstalled during the upgrade process, the IBM 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 279 before you start the upgrade process. The result of not completing these steps can be a non-functional server.

Alternate Method: Move your Worklight apps and data to a new Liberty server

This alternate upgrade method migrates your Worklight Console and Application Center 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.
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 "Worklight Server installation process walkthrough" on page 53.
4. Configure the Liberty server for your production environment.
5. Modify the Ant files created in section "Identify the Worklight WAR file and prepare the Ant deployment script" on page 229 to point to the newly installed WebSphere Application Server Liberty Core.
6. When you reach the step "Running IBM Installation Manager and completing the Application Center upgrade" on page 243, follow the instructions for "Upgrading from Worklight Server V5.0.6.x (changing the Liberty server)" on page 245.

Become familiar with IBM Installation Manager before you start:

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.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Running	Running

Procedure

Before you use IBM Installation Manager to upgrade your production server, familiarize (or re-familiarize) yourself with how it works:

1. Make sure that you have the appropriate version of IBM Installation Manager installed on the installation workstation.

   Note: IBM Installation Manager is sometimes referred to on the eXtreme Leverage and Passport Advantage sites and on the distribution disks as *IBM*

Rational Enterprise Deployment. The filenames for the images take the form `IBM Rational Enterprise Deployment <version number><hardware platform> <language>`; for example, `IBM Rational Enterprise Deployment V1.6.3.1Windows Multilingual`.

Use IBM Installation Manager V1.6.3.1 or later, especially on Windows. For more information about IBM Installation Manager procedures, see the IBM Installation Manager user documentation.

Important: If you are performing an in-place upgrade, and you have IBM Installation Manager installed on your computer in several different modes, for example, administrator mode and nonadministrator (single user) mode, you must use the same mode used to install the previous version of Worklight Server.

2. 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.”
3. Verify that the products that you want to update are contained in the IBM Installation Manager repositories.
4. If you do not plan to use IBM Installation Manager in graphical mode but in silent install mode, review the procedures for a silent install as documented in “Silent installation” on page 65 and “Working with sample response files for IBM Installation Manager” on page 66 and prepare your response file.

   To prepare your response file from sample response files, create a response file based on the following:

   Table 48. Sample upgrade response files in the `Silent_Install_Sample_Files.zip`

Initial version of Worklight Server	Sample file
V5.x	`upgrade-initially-v5.xml`
V6.x	`upgrade-initially-v6.xml`

   In the `<offering>` element in the `<install>` element, set the `version` attribute to match the release you want to upgrade to, or remove the `version` attribute if you want to upgrade to the newest version available in the repositories.

Information about the repositories

There are three types of repositories: base repositories, delta repositories, and interim fix 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 of the previous release version to be functional.
- An *interim fix repository* is an installation package that is available from FixCentral and is labeled as an interim fix, and that is only versioned by a build number. It requires the repositories of the previous release version to be functional: either a base repository, or both a base repository and a delta repository.

To install a major release (for example, Worklight Server V6.1.0), you need only:

- The base repository V6.1.0 installation package from Passport Advantage or physical media.

To install a fix pack release (for example, Worklight Server V6.1.0.1), you need:
- The corresponding base repository (such as Worklight Server V6.1.0) installation package from Passport Advantage or physical media. The corresponding base repository for V6.1.0.x fixpacks is the V6.1.0 release.
- The appropriate V6.1.0.x installation package from FixCentral.

For a fix pack installation, you must add both repositories to the list known to IBM Installation Manager. Then, in the example given, IBM Installation Manager recognizes the V6.1.0 release as an **Install** choice and the V6.1.0.x release (or interim fix) 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.

Review of the basic IBM Installation Manager steps

CAUTION:
The following steps are not the actual installation. They are preparatory tasks to help you ensure that you have everything that is required for the upgrade. Be sure to click Cancel in the last step.

1. Start IBM Installation Manager.
2. 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.
3. Click **Install**.
4. Verify that the products list contains everything that you need.
5. Click **Cancel**. Do not proceed with the installation.

Starting the Worklight Server V6.1.0 upgrade process

In this phase of the upgrade process, you shut down and back up the application server and IBM Worklight databases and perform other pre-installation tasks.

When you finish the tasks that are listed in “Preparing for the upgrade to Worklight Server 6.1.0” on page 228, you can begin the actual upgrade process.

Note: After you complete this phase of the upgrade process, your Worklight Server, Application Center, database, and application server are (or can be) 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.

Verify the ownership of your Worklight Server files:

Before you begin the actual installation, check the ownership of all Worklight Server files.

About this task

The upcoming step “Running IBM Installation Manager and completing the Application Center upgrade” on page 243 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.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Running	Running

Procedure

If you previously installed Worklight Server with the nonadministrator (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 and nonadministrator modes, see Administrator, nonadministrator, and group mode. Note that group mode is not supported for Worklight Server installation.

On UNIX, you can list the files and directories that do not fulfill this condition with the following command:

```
cd WL_INSTALL_DIR
find . '!' -user "$USER" -print
```

This command should produce no output.

Back up your application server:

Back up the directory that contains the application server and its configuration.

About this task

Back up your application server so that you can recover in case of an unsuccessful server upgrade. 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.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Running	Running

Procedure

Back up all the application servers (or network deployment nodes) where the Application Center application or a Worklight project's WAR file is installed.

For WebSphere Application Server Liberty Profile:

- Back up the `usr` directory. By default this directory is located in `<LibertyInstallDir>/usr`, but its location can be redefined by the `WLP_USER_DIR` variable in `<LibertyInstallDir>/env/server.env`.

For WebSphere Application Server Full Profile:

- If your original installation was to one or more servers under the control of a deployment manager, and not a single stand-alone server:
  - Either use the WebSphere backupConfig command to back up the deployment manager node.
  - Or back up the `config` directory inside the deployment manager profile directory.
- If your original installation was to a stand-alone server:
  - Either use the WebSphere backupConfig command to back up the entire node.
  - Or back up the application server profile directory.

See the documentation for Apache Tomcat to determine the directories to back up for this application server.

Shutting down the application server:

If you use WebSphere Application Server Liberty Profile or Apache Tomcat, you must shut the application server down during this step.

About this task

You must shut down the application server before running IBM Installation Manager in the following three cases:

- If your application server is Apache Tomcat.
- If your application server is WebSphere Application Server Liberty Core.
- If your application server is the embedded version of WebSphere Application Server Liberty Profile installed by the Worklight Server V5.0.6 or earlier installer.
  - In this case, you must also shut down all processes that have either their current working directory inside or opened files inside the Worklight installation directory hierarchy.
  - On Windows, you must also shut down all such processes inside the Liberty Worklight Server directory hierarchy, located in `C:\ProgramData\IBM\Worklight\WAS85liberty-server`.

Otherwise, if the application server is running when IBM Installation Manager starts the upgrade, some upgrade operations may fail, leaving the Worklight Server installation in an inconsistent state.

Is this step required for your upgrade path?				**System status after this step, if both Application Center and Worklight Server are on the same application server**	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Stopped (Liberty and Tomcat) Running (others)	Stopped (Liberty and Tomcat) Running (others)

Procedure

For Apache Tomcat and WebSphere Application Server Liberty Core, use the administration commands to shut down the application server as you would normally.

For the embedded version of WebSphere Application Server Liberty Server, you can use the following 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/
   ```

Stop all instances of the Application Center applications:

Stop the applications currently running on Application Center.

About this task

If you have installed Application Center on multiple servers, networked or not, then all instances of the IBM Application Center Console and IBM Application

Center Services must be stopped before you run IBM Installation Manager to upgrade the Worklight Server Installation.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes (if installed on multiple servers)	Yes (if installed on multiple servers)	See fix pack installation instructions	See interim fix installation instructions	Stopped (all instances)	Stopped (Liberty and Tomcat) Running (others)

Procedure

The reason this step is required is that IBM Installation Manager migrates the schema of the database so that it can be used with Worklight Server V6.1.0. No instance of Application Center can be running while this operation is performed.

After the database is migrated, only migrated Application Center applications should be run, because only migrated applications are able to read and write to the new databases. Otherwise, the Application Center database may be corrupted.

If you have installed Application Center only once, this operation will be done automatically by IBM Installation Manager.

Back up the Application Center database:

Before you run IBM Installation Manager to install Worklight Server V6.1.0, back up your Application Center database.

About this task

In the upgrade process, the Application Center database is updated and migrated to a schema compatible with Worklight Server V6.1.0. This operation can not be undone. If, for any reason, you decide to rollback the upgrade of Worklight Server, you will need this backup.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	Stopped (all instances)	Stopped (Liberty and Tomcat) Running (others)

Procedure

Use the standard procedures for your DBMS (IBM DB2, Oracle, or MySQL) to back up the Application Center database. The default name for the Application Center database, unless you modified it at install time, is as follows:

- For IBM DB2, MySQL, and Oracle, if you installed Worklight V5.0.6: `APPCNTR`
- For IBM DB2 and MySQL if you installed Worklight V6.0.0 or later: `APPCNTR`
- For Oracle, if you installed Worklight V6.0.0 or later: `ORCL`

.

The Worklight and Worklight Reports databases are backed up as well, but in a later step of this procedure. For more information, see step "Back up the Worklight and Worklight Reports databases" on page 249 of this upgrade procedure.

Running IBM Installation Manager and completing the Application Center 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 "Preparing for the upgrade to Worklight Server 6.1.0" on page 228 and "Starting the Worklight Server V6.1.0 upgrade process" on page 238 sections that preceded this step.

It is also possible to run IBM Installation Manager in silent install mode, using response files either generated by using it in wizard mode on a machine where a GUI is available, or by working with sample response files supplied with IBM Worklight. For more information, see "Silent installation" on page 65 and "Working with sample response files for IBM Installation Manager" on page 66.

Upgrading from Worklight Server V6.x:

In this step, you run IBM Installation Manager to perform the actual upgrade from Worklight Server V6.x to Worklight Server V6.1.0.

About this task

IBM Installation Manager completes the following tasks:

- It installs on your disk the files and tools that are required to deploy IBM Worklight on your application 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 V6.1.0. To see a copy of the upgrade scripts, you can install 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.
  - Configures the application server for running the Application Center.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
No	Yes	Yes	Yes	Stopped (all instances)	Stopped (Liberty and Tomcat) Running (others)

Procedure

Having prepared for this step by becoming familiar with IBM Installation Manager and ensuring that you have all the proper repositories for the upgrade, start the actual installation using the following steps:

1. Start IBM Installation Manager.
2. Click **Update**.
3. Step through the installation wizard, following the onscreen prompts to complete the upgrade.

Upgrading from Worklight Server V5.0.6.x:

Use this procedure to upgrade from Worklight Server V5.0.6.x to Worklight Server V6.1.0 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.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes (unless Liberty server was installed by Worklight Server V5.0.6)	No	No	No	Stopped (all instances)	Uninstalled

Procedure

1. Start IBM Installation Manager.

2. Click **Install**. The package name for IBM Worklight Server has changed between Worklight Server V5.x and V6.1.0, 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 to Worklight Server V6.1.0 and rolling upgrade” on page 234), 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 bythe current version of Worklight Server. 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.

Upgrading from Worklight Server V5.0.6.x (changing the Liberty server):

This step contains special instructions if you are migrating to a new instance of WebSphere Application Server Liberty Profile.

About this task

This task is part of the “Alternate Method: Move your Worklight apps and data to a new Liberty server” on page 236 section of the “Packaging change of WebSphere Application Server Liberty Profile in IBM Worklight V6.x” on page 235 step.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes (if Liberty server was installed by Worklight Server V5.0.6)	No	No	No	Stopped (all instances)	Uninistalled

Procedure

Having prepared for this step by becoming familiar with IBM Installation Manager and ensuring that you have all the proper repositories for the upgrade, start the actual installation using the following steps:

1. Start IBM Installation Manager.
2. Click **Install**.
3. Select a new package group.
4. Step through the installation wizard. Enter the database settings used to install Application Center for version V5.0.6.
5. For the Application Server choice, select the newly installed WebSphere Application Server Liberty Core.

Restore the Application Center configurations and restart the application server:

In this step you restore the required configurations of Application Center that you made note of in a previous step.

About this task

Restore the configurations that you previously identified in step "Review and note the Application Server configuration for Worklight Server and Application Center" on page 231.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	Upgraded	Running (if upgrading from V6.0.x), Stopped (if upgrading from V5.0.6.x)

Procedure

1. The applications to restore are as follows:
   - For the applications:
     - The Application Center Console
     - The Application Center Services
2. The JDBC data sources to restore are as follows:
   - The Application Center database
3. When you have restored these configurations, restart the application server that was upgraded.

Results

At the end of this step, Application Center is upgraded. All applications previously loaded in Application Center should be available.

However, if this Application Center is running on the same application server as a Worklight Console, that application server is shut down again in a later step, and is only restarted in subsequent steps.

Upgrading the Worklight Console for Worklight Server 6.1.0

In these post-installation steps, you set or restore configurations for Worklight Server, its databases, and Worklight Console, and restart the application server.

Since IBM Worklight V6.0.0, it is possible to deploy several project WAR files to an application server. You must perform these steps for each Worklight project WAR file that you deployed and that you want to upgrade to V6.1.0. If you migrated a project WAR file and deployed it on multiple application servers, all instances must be upgraded.

Complete each of the following steps, as required for your particular upgrade path.

Stop all Worklight Server instances:

Before you complete the next upgrade steps, you must shut down all instances of Worklight Console. You must also disable the auto start mode of the Worklight Console if you upgrade from IBM Worklight V6.0.0.x on WebSphere Application Server full profile.

About this task

If you installed Worklight Server on multiple servers, networked or not, you must stop all instances of the Worklight Console before you perform the next steps.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	Upgraded	Stopped (all instances)

Procedure

The reason this step is required is that in the next steps you migrate the schema of the database so that it can be used with Worklight Server V6.1.0. No instance of Worklight Console can be running while this operation is performed.

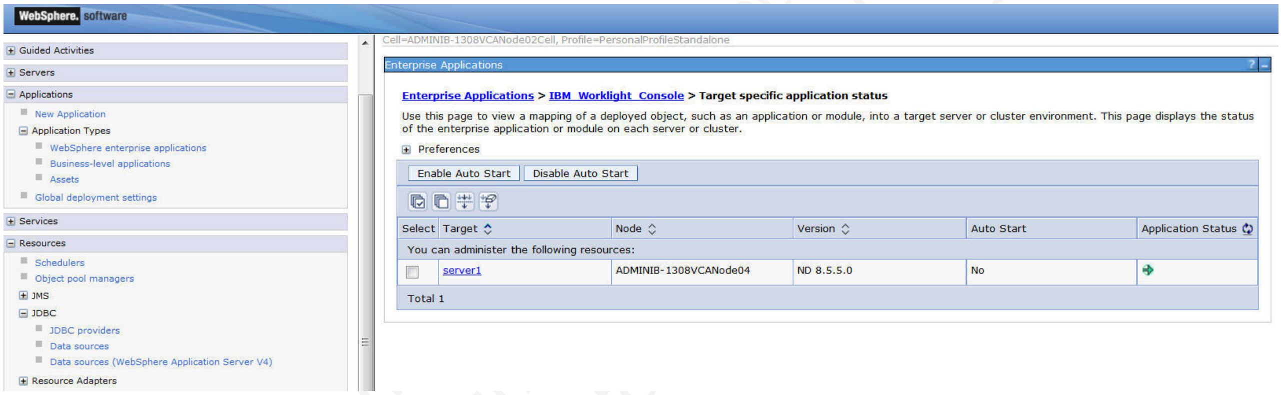
After the database is migrated, only migrated Worklight Console applications should be run, because only migrated applications are able to read and write to the new databases. Otherwise, the Worklight Console database might be corrupted.

You must perform this operation even if you installed Worklight Console only once.

Important: In addition to stopping the Worklight Console, if you upgrade from IBM Worklight V6.0.0.x on WebSphere Application Server full profile, you must disable the auto start mode of the Worklight Console application during the upgrade, before you shut down the Worklight Server. If the auto start mode is not disabled, the Worklight Console modifies the database when the server is started in step “Upgrading the Worklight Console for Worklight Server 6.1.0” on page 247 and prevents the new project WAR file from starting.

To disable the auto start mode:

1. Log in to the WebSphere Console.
2. Go to the menu **Applications** > **Application Types** > **WebSphere enterprise applications**, and list the applications.
3. In the table, select the Worklight Console application, whose default name is **IBM_Worklight_Console**.
4. In **Detail Properties** click on **Target Specific Application Status**.
5. Select all the target servers, or the cluster where the application is installed.
6. Click **Disable Auto Start**.
7. Click **Save** to save the configuration.
8. Verify that the **Auto Start** property in the table is set to **No**.



Cell=ADMINIB-1308VCANode02Cell, Profile=PersonalProfileStandalone

Enterprise Applications > IBM_Worklight_Console > Target specific application status

Use this page to view a mapping of a deployed object, such as an application or module, into a target server or cluster environment. This page displays the status of the enterprise application or module on each server or cluster.

Select	Target	Node	Version	Auto Start	Application Status
You can administer the following resources:					
□	server1	ADMINIB-1308VCANode04	ND 8.5.5.0	No	➜
Total 1					

Figure 20.

Shutting down the application server to be upgraded:

For certain configurations, in this step you shut down the application server prior to completing subsequent steps.

About this task

For certain types of application servers (see the table and “Procedure” on page 249 section below), you must shut down the application server before proceeding to subsequent steps.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	V6.0.0.x to V6.1.0	V6.1.0 to V6.1.0.x (Fix Pack)	V6.1.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)	Stopped (all instances)

Procedure

In the following cases, you must shut down the application server before you undeploy applications from the Worklight Console application:

- If the application server is WebSphere Application Server Liberty Profile and the OS is Windows.
- If the application server is Apache Tomcat, and the OS is Windows or the database type is Apache Derby.

If these application servers are not shut down, the undeploy operations may fail.

Back up the Worklight and Worklight Reports databases:

Back up the contents of your IBM Worklight project databases.

About this task

Important: Before performing this step, verify that you have completed step “Stop all Worklight Server instances” on page 247, and that no instance of Worklight Server is still running, and thus still using these databases.

During the upgrade process in the next step, the Worklight and the Worklight Reports databases are migrated to a schema compatible with Worklight Server V6.1.0. This operation can not be undone.

If, for some reason, you decide to rollback the upgrade of Worklight Server, you will need this backup.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)	Stopped (all instances)

Procedure

The default names for the databases, unless you modified them at install time, are as follows:

- For IBM DB2, Derby, MySQL, and Oracle, if you installed IBM Worklight V5.0.6.x: `WRKLGHT` and `WLREPORT`
- For IBM DB2, Derby, and MySQL, if you installed IBM Worklight V6.x: `WRKLGHT` and `WLREPORT`
- For Oracle, if you installed IBM Worklight V6.x, for Oracle: `ORCL`

Upgrade the Worklight and Worklight Reports databases:

In this step you upgrade the schemas of your IBM Worklight project databases.

About this task

Important: Before performing this step, verify that you have completed step “Stop all Worklight Server instances” on page 247, and that no instance of Worklight Server is still running, and thus still using these databases.

In this step, you run the Ant scripts to upgrade the schemas of your Worklight and Worklight Reports databases.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)	Stopped (all instances)

Procedure

1. If you upgrade from IBM Worklight V6.0.0.x, and the application server is WebSphere Application Server full profile, make sure you disabled the auto start mode for all instances of the Worklight Console application, as specified in "Stop all Worklight Server instances" on page 247.
2. Locate the Ant file that you created in section "Identify the Worklight WAR file and prepare the Ant deployment script" on page 229.
3. Verify that the `taskdef`for the `worklight-ant-deployer.jar` uses the correct directory containing the upgraded installation of Worklight Server V6.1.0.

   In the example shown below, you need to check the value of the property `worklight.server.install.dir` because this property is used to define the directory of the `worklight-ant-deployer.jar` in the `taskdef` tag:

   ```
   <property name="worklight.server.install.dir" value="c:/Program File/IBM/Worklight"/>

     [...]

     <taskdef resource="com/worklight/ant/defaults.properties">
       <classpath>
         <fileset dir="${worklight.server.install.dir}/WorklightServer">
           <include name="worklight-ant-deployer.jar"/>
         </fileset>
       </classpath>
     </taskdef>
   ```

   This verification step is extremely important. It defines the version of IBM Worklight that you use to migrate the databases, to deploy the WAR file, and to install the Worklight runtime library for the Worklight Console.
4. Set the environment variable `ANT_HOME` to `<WORKLIGHT_INSTALL_DIR>/tools/apache-ant-1.8.4/`.

   This is the version of Apache Ant for which the Worklight deployment scripts have been tested. If you do not set this environment variable before running the script, and have another installation of Ant on your computer, that installation may be used.
5. Launch the databases target of the Ant file with this command:

   ```
   <WORKLIGHT_INSTALL_DIR>/tools/apache-ant-1.8.4/bin/ant -f <your file> databases
   ```

   Note: If you created an Ant file with your own target names, the Ant task to launch is `configuredatabase`.

This procedure upgrades the database schemas for the Worklight and Worklight Reports databases to version V6.1.0.

Upgrade the Worklight Server console:

In this step you run the Ant script to upgrade Worklight Console to version 6.1.0.

About this task

In this step, you run the same Ant script as in the previous step, but with a different parameter to indicate the Ant target and the nature of the upgrade.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Partially Upgraded (other cases)

Procedure

1. Locate the Ant file that you created in section "Identify the Worklight WAR file and prepare the Ant deployment script" on page 229.
2. Verify that the `taskdef` for the `worklight-ant-deployer.jar` uses the correct directory containing the upgraded installation of Worklight Server V6.1.0.

   In the example shown below, you need to check the value of the property `worklight.server.install.dir` because this property is used to define the directory of the `worklight-ant-deployer.jar` in the `taskdef` tag:

   ```
   <property name="Worklight.server.install.dir" value="c:/Program File/IBM/Worklight"/>

     [...]

     <taskdef resource="com/worklight/ant/defaults.properties">
       <classpath>
         <fileset dir="${worklight.server.install.dir}/WorklightServer">
           <include name="worklight-ant-deployer.jar"/>
         </fileset>
       </classpath>
     </taskdef>
   ```

   This verification step is extremely important. It defines the version of IBM Worklight that you use to migrate the databases, to deploy the WAR file, and to install the Worklight runtime library for the Worklight Console.
3. Set the environment variable `ANT_HOME` to <*WORKLIGHT_INSTALL_DIR*>`/tools/apache-ant-1.8.4/`.

This is the version of Apache Ant for which the Worklight deployment scripts have been tested. If you do not set this environment variable before running the script, and have another installation of Ant on your computer, that installation may be used.

4. Select the Ant target to use:
   - To upgrade from V5.0.6.x, use: **install**
   - To upgrade from V6.0.0.x, use: **minimal-update**
   - To upgrade from V6.1.0 to a fix pack or interim fix release, use:
     - Either **uninstall**, then **install** or **minimal-update**
     - Or **minimal-update**

   This choice depends on the nature of the changes in the fix. For more information, see the fix pack or interim fix installation instructions.
5. If you upgrade from IBM Worklight V6.0.0.x, and the application server is WebSphere Application Server full profile, enable the auto start mode for the Worklight Console, which was disabled in “Stop all Worklight Server instances” on page 247. If you do not enable it, the application needs to be started explicitly after the server starts.
6. Run Ant with the selected target:

   ```
   <WORKLIGHT_INSTALL_DIR>/tools/apache-ant-1.8.4/bin/ant -f <your file> <target defined in step
   ```

   This script:
   - Migrates the WAR file to match the runtime of the Worklight Server installation.
   - Installs the Worklight Console and its runtime to the application server, with the root context defined in the Ant file (or /worklight by default).
   - Connects the new console to the database containing the applications (.wlapp files) and adapters deployed to the Worklight Console of the previous version (V5.0.6.x or V6.0.0.x).

Restore the Worklight Server Configuration:

In this step you restore the required configurations of Worklight Server that you made note of in a previous step.

About this task

Restore the configurations that you previously identified in step “Review and note the Application Server configuration for Worklight Server and Application Center” on page 231.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	No	See fix pack installation instructions	See interim fix installation instructions	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)	Stopped (embedded Liberty, Liberty on Windows, Tomcat on Windows), Upgraded (other cases)

Procedure

1. The applications to restore are as follows:
   - For the applications:
     - The Worklight Console
   - For the JDBC data source:
     - The Worklight database

Restart the application server:

In this final step, you restart the application server.

About this task

Now that the upgrade of Worklight Server is completed, restart your application server.

Is this step required for your upgrade path?				System status after this step, if both Application Center and Worklight Server are on the same application server	
V5.0.6.x to V6.1.0	**V6.0.0.x to V6.1.0**	**V6.1.0 to V6.1.0.x (Fix Pack)**	**V6.1.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes	Yes	Yes	Upgraded	Upgraded

Procedure

1. Use your standard procedures to start the application server, or restart the application server if it was running in this step, so that all changes are taken into account.

At the end of this step, the Worklight Console is upgraded. All applications that you previously deployed should be available, along with their environments (those that are supported by Worklight Server V6.1.0).

If you upgraded from Worklight V5.0.6.x, the URL of the Worklight Console has changed. If you did not specify a context root in the Ant file, its context root is `/worklight`.

Additional Worklight Server V6.1.0 upgrade information

This section contains additional information that may be of use if you have additional test or pre-production databases that must be updated, if you need to update HTTP redirections on networked servers, if you are manually upgrading the application server, or in the event of a failed upgrade.

Recovering from an unsuccessful Worklight Server V6.1.0 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 IBM Installation Manager.
2. Install the old version of Worklight Server with IBM Installation Manager, specifying the same installation parameters that you used previously.
3. Restore the databases. For more information, see "Back up the Worklight and Worklight Reports databases" on page 249
4. Restore the application server. For more information, see "Back up your application server" on page 239.
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`.

Manually upgrading the Worklight Server V6.1.0 databases:

Follow these instructions to manually update the Worklight project 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, if you have test or pre-production databases as part of your production environment, each served by a different Worklight database or schema, you can use this procedure to update their schemas.

Updating these alternative databases is done by running a sequence of database scripts. The upgrade scripts are contained in the just-installed (version 6.1.0) Worklight Server directory.

Scripts for DB2

For an upgrade from Worklight Server V5.0.6.x 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)

For an upgrade from Worklight Server V6.0.0.x to V6.1.0:

- WorklightServer/databases/upgrade-worklight-60-61-db2.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-60-61-db2.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-60-61-db2.sql (for APPCNTR)

These scripts are applied similarly to steps 4 and 6 in “Setting up your DB2 databases manually” on page 732

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 V5.0.6 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)

For an upgrade from Worklight Server V6.0.0.x to V6.1.0:

- WorklightServer/databases/upgrade-worklight-60-61-mysql.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-60-61-mysql.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-60-61-mysql.sql (for APPCNTR)

These scripts are applied similarly to step 1.b in “Setting up your MySQL databases manually” on page 741.

Scripts for Oracle

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)

For an upgrade from Worklight Server V6.0.0.x to V6.1.0:

- WorklightServer/databases/upgrade-worklight-60-61-oracle.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-60-61-oracle.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-60-61-oracle.sql (for APPCNTR)

These scripts are applied similarly to step 3 in “Setting up your Oracle databases manually” on page 744.

Manually upgrading the application server:

Follow these instructions to manually upgrade the application server.

The recommended way to upgrade Worklight Server is to use IBM Installation Manager, either in its graphical mode or in silent mode with a response file, and the Ant tasks, as described previously.

However, if this is not applicable to your installation and you must update your application server manually, use a different series of steps.

Instead of completing the tasks “Running IBM Installation Manager and completing the Application Center upgrade” on page 243 and “Upgrade the Worklight Server console” on page 252, use the following procedure:

- Upgrade the databases manually as specified in section “Manually upgrading the Worklight Server V6.1.0 databases” on page 255.
- Review the manual installation procedures at:
  - “Manually installing Application Center” on page 82
  - “Deploying a project WAR file and configuring the application server manually” on page 767
- Update the items manually. This includes, at a minimum:
  - The WAR file for the Application Center console, Application Center services, and the Worklight Console.
  - The Worklight library worklight-jee-library.jar.
  - The Worklight project's WAR file, which must be migrated to the current version of the server using the migrate Ant task described at “Migrating a project WAR file for use with a new Worklight Server” on page 767.

Verifying and updating the HTTP redirections for Worklight Server V6.1.0:

If you are upgrading Worklight Server on a clustered application server environment, you should also update IBM HTTP Server after you install IBM Worklight V6.1.0.

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 V6.1.0. 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 180
- “Setting up IBM Worklight in an IBM WebSphere Application Server Liberty Profile farm” on page 191

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>
```

Upgrading from Worklight Server V6.1.0 to V6.1.0.x in a production environment

Some upgrade steps are optional when you upgrade to a fix pack. You must follow specific instructions when you upgrade from Worklight Server V6.1.0 to V6.1.0.x.

To upgrade from Worklight Server V6.1.0 to V6.1.0.x (fix pack) in a production environment, you must follow the steps that are listed in the topic “Upgrading to Worklight Server V6.1.0 in a production environment” on page 226, but some of these steps are optional for a fix pack. You can see in *Table 1* the status of these optional steps for the upgrade to Worklight Server V6.1.0.x.

Table 49. Status of the optional steps for upgrading to the V6.1.0.x fix pack

Upgrade step	Is this step required for V6.1.0.x fix pack?
“Review and note the Application Server configuration for Worklight Server and Application Center” on page 231	Not required
“Stop all instances of the Application Center applications” on page 241	Required
“Back up the Application Center database” on page 242	Not required
“Restore the Application Center configurations and restart the application server” on page 246	Only restart is required
“Stop all Worklight Server instances” on page 247	Required
“Back up the Worklight and Worklight Reports databases” on page 249	Not required
“Upgrade the Worklight and Worklight Reports databases” on page 250	Not required
“Upgrade the Worklight Server console” on page 252	Use the Ant target **`minimal-update`** (step 4).
“Restore the Worklight Server Configuration” on page 253	Not required

Upgrading from Worklight Server V6.1.0 to an interim fix in a production environment

Some upgrade steps are optional when you upgrade to an interim fix. You must follow specific instructions when you upgrade from Worklight Server V6.1.0 to an interim fix.

To upgrade from Worklight Server V6.1.0 to an interim fix in a production environment, you must follow the steps that are listed in the topic “Upgrading to Worklight Server V6.1.0 in a production environment” on page 226, but some of these steps are optional for an interim fix. You can see in *Table 1* the status of these optional steps for the upgrade to Worklight Server interim fix.

Table 50. Status of the optional steps for upgrading to an interim fix

Upgrade step	Is this step required for an interim fix?
“Review and note the Application Server configuration for Worklight Server and Application Center” on page 231	Required
“Stop all instances of the Application Center applications” on page 241	Required
“Back up the Application Center database” on page 242	Not required
“Restore the Application Center configurations and restart the application server” on page 246	Required
“Stop all Worklight Server instances” on page 247	Required
“Back up the Worklight and Worklight Reports databases” on page 249	Not required
“Upgrade the Worklight and Worklight Reports databases” on page 250	Not required
“Upgrade the Worklight Server console” on page 252	Use the Ant target `minimal-update` (step 4).
“Restore the Worklight Server Configuration” on page 253	Not required

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 303, and “Migrating from IBM Worklight V5.0.0.3 to V5.0.5” on page 308.

Note: You can migrate a workspace and its associated projects, but the upgrade of Worklight Studio to V6.0.0 requires a new installation (see the following section, “Worklight Studio” on page 260).

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 268.
- To migrate existing projects to a new instance of Worklight Studio, see “Migrating projects to a new Worklight Studio instance” on page 269.

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 270.

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 “Migrating your Worklight projects” on page 271.
- **Migration of projects that use 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 215.

- **Usage of WL.App.close**: 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 the WL.App.close API description under “JavaScript client-side API” on page 696.

Third-party libraries

Cordova: IBM Worklight V6.0.0 is now based on Cordova 2.6. Compared to the earlier version, Cordova 2.6 includes new, modified, deprecated, and removed items. 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 262.

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 267.

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:

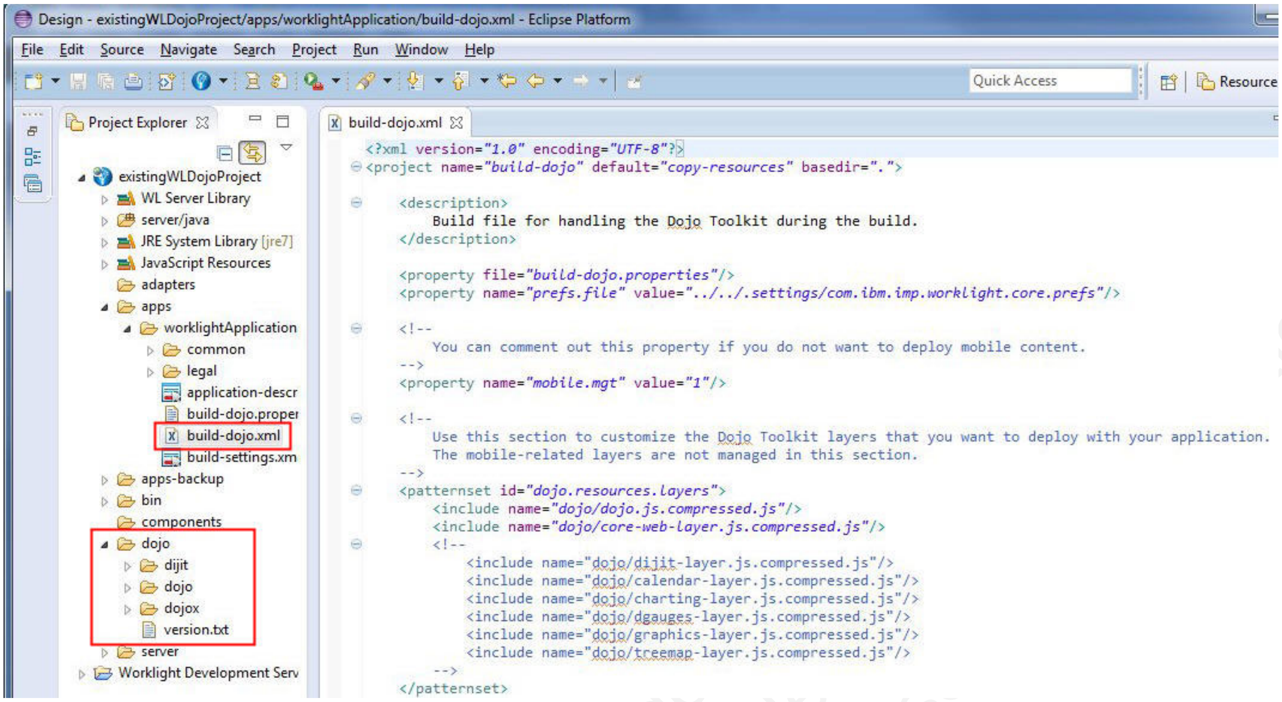
```
<widget>
 <plugins>
  <!--Cordova Plugins-->
  <!--Worklight Plugins-->
  <plugin name=your plugin/>
 </plugins>
</widget>
```

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-dojo.xml` file that controlled which pieces of Dojo were built into the IBM Worklight application, as shown in Figure 21 on page 263.



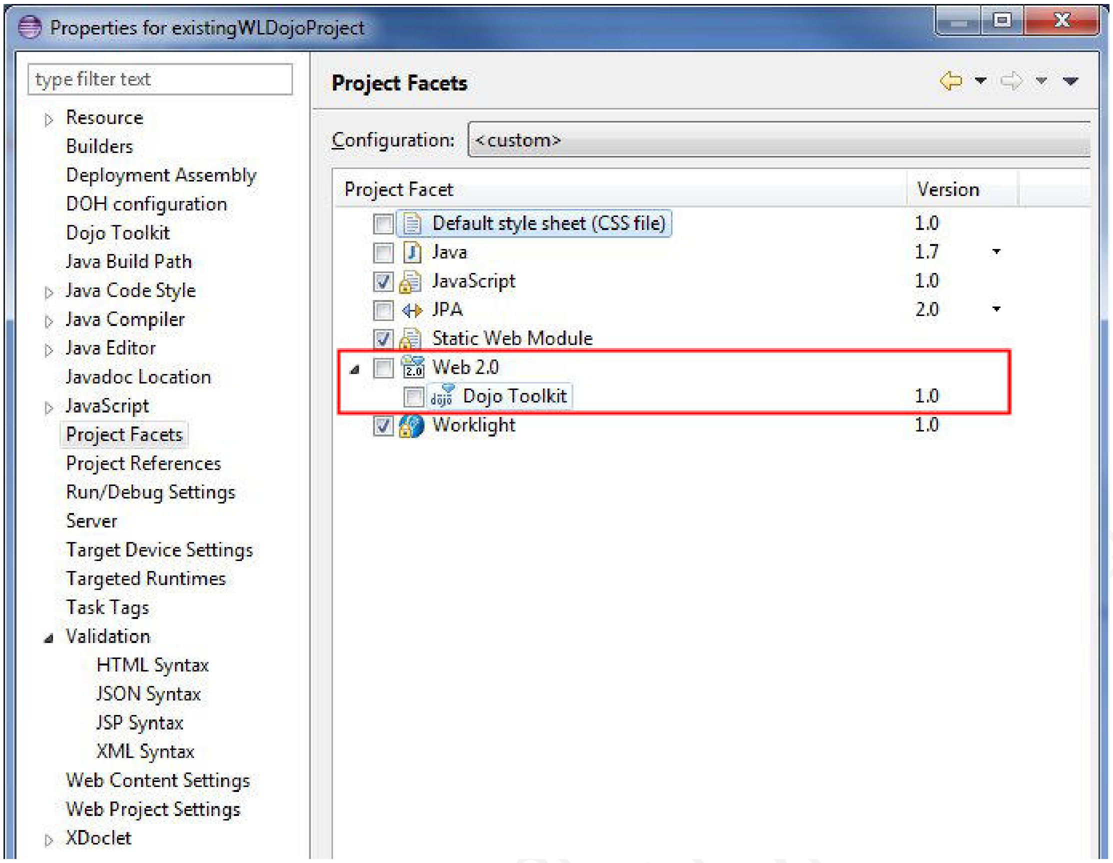
```
<?xml version="1.0" encoding="UTF-8"?>
<project name="build-dojo" default="copy-resources" basedir=".">

    <description>
        Build file for handling the Dojo Toolkit during the build.
    </description>

    <property file="build-dojo.properties"/>
    <property name="prefs.file" value="../../.settings/com.ibm.imp.worklight.core.prefs"/>

    <!--
        You can comment out this property if you do not want to deploy mobile content.
    -->
    <property name="mobile.mgt" value="1"/>

    <!--
        Use this section to customize the Dojo Toolkit layers that you want to deploy with your application.
        The mobile-related layers are not managed in this section.
    -->
    <patternset id="dojo.resources.layers">
        <include name="dojo/dojo.js.compressed.js"/>
        <include name="dojo/core-web-layer.js.compressed.js"/>
        <!--
            <include name="dojo/dijit-layer.js.compressed.js"/>
            <include name="dojo/calendar-layer.js.compressed.js"/>
            <include name="dojo/charting-layer.js.compressed.js"/>
            <include name="dojo/dgauges-layer.js.compressed.js"/>
            <include name="dojo/graphics-layer.js.compressed.js"/>
            <include name="dojo/treemap-layer.js.compressed.js"/>
        -->
    </patternset>
```

Figure 21. 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 “Working with the Dojo Library Project that serves Dojo resources” on page 358). 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.



Properties for existingWLDojoProject

type filter text

- Resource
- Builders
- Deployment Assembly
- DOH configuration
- Dojo Toolkit
- Java Build Path
- Java Code Style
- Java Compiler
- Java Editor
- Javadoc Location
- JavaScript
- Project Facets
- Project References
- Run/Debug Settings
- Server
- Target Device Settings
- Targeted Runtimes
- Task Tags
- Validation
  - HTML Syntax
  - JSON Syntax
  - JSP Syntax
  - XML Syntax
- Web Content Settings
- Web Project Settings
- XDoclet

Project Facets

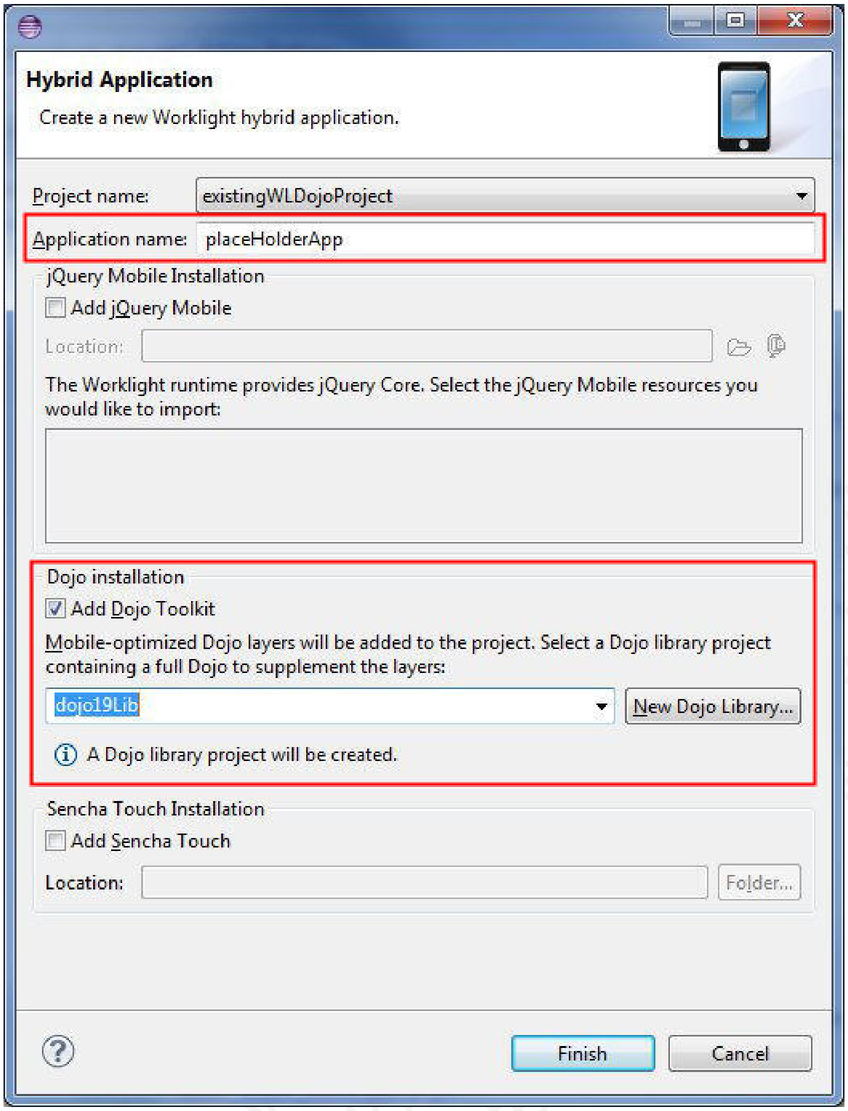
Configuration: <custom>

Project Facet	Version
□ Default style sheet (CSS file)	1.0
□ Java	1.7
☑ JavaScript	1.0
□ JPA	2.0
☑ Static Web Module	
□ Web 2.0	
□ Dojo Toolkit	1.0
☑ Worklight	1.0

Figure 22. 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**.



Hybrid Application

Create a new Worklight hybrid application.

Project name: existingWLDojoProject

Application name: placeHolderApp

jQuery Mobile Installation

□ Add jQuery Mobile

Location:

The Worklight runtime provides jQuery Core. Select the jQuery Mobile resources you would like to import:

Dojo installation

☑ Add Dojo Toolkit

Mobile-optimized Dojo layers will be added to the project. Select a Dojo library project containing a full Dojo to supplement the layers:

dojo19Lib

A Dojo library project will be created.

Sencha Touch Installation

□ Add Sencha Touch

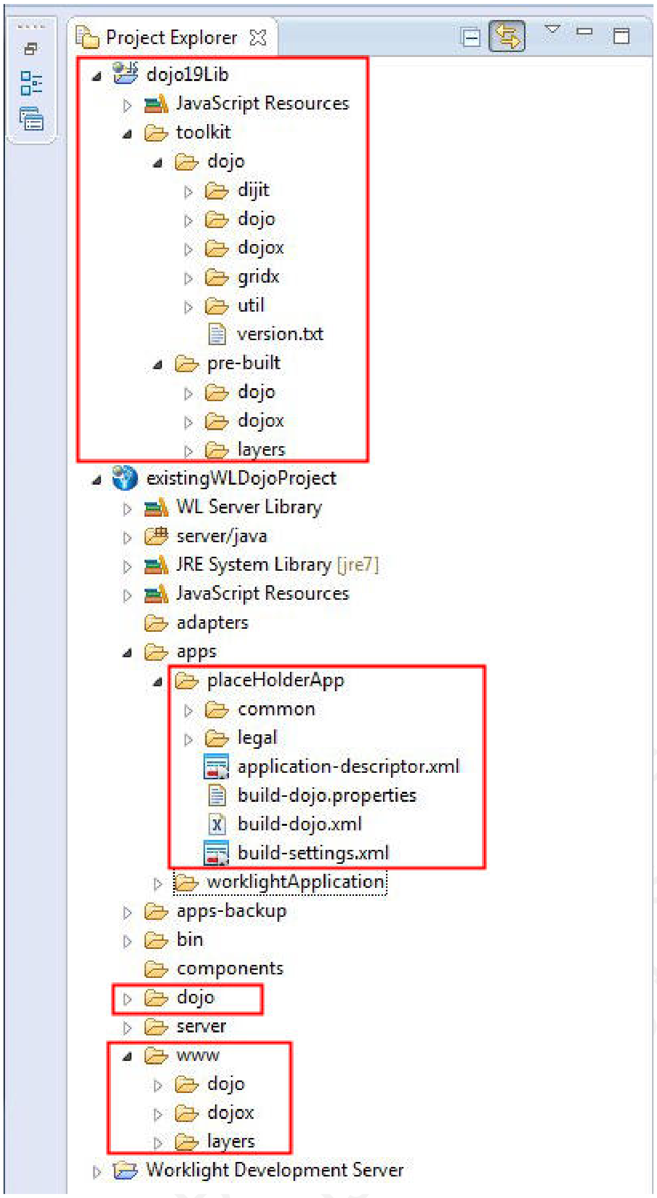
Location:

Figure 23. Adding Dojo toolkit to an application

8. Name and configure the Dojo Library. For more information, see “Working with the Dojo Library Project that serves Dojo resources” on page 358.
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 24 on page 266 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)



Project Explorer

- dojo19Lib
  - JavaScript Resources
  - toolkit
    - dojo
      - dijit
      - dojo
      - dojox
      - gridx
      - util
      - version.txt
    - pre-built
      - dojo
      - dojox
      - layers
- existingWLDojoProject
  - WL Server Library
  - server/java
  - JRE System Library [jre7]
  - JavaScript Resources
  - adapters
  - apps
    - placeHolderApp
      - common
      - legal
      - application-descriptor.xml
      - build-dojo.properties
      - build-dojo.xml
      - build-settings.xml
    - worklightApplication
  - apps-backup
  - bin
  - components
  - dojo
  - server
  - www
    - dojo
    - dojox
    - layers
- Worklight Development Server

Figure 24. New application in an existing Dojo project

10. Copy the `build-dojo.xml` and `build-dojo.properties` files from the placeholder application to the application you want to migrate.

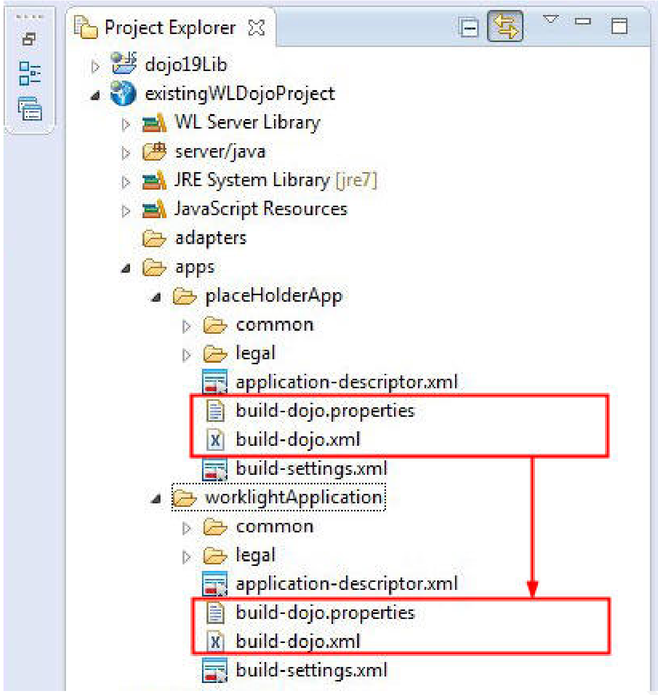


Figure 25. Copy of the `build-dojo.xml` and `build-dojo.properties` files

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.

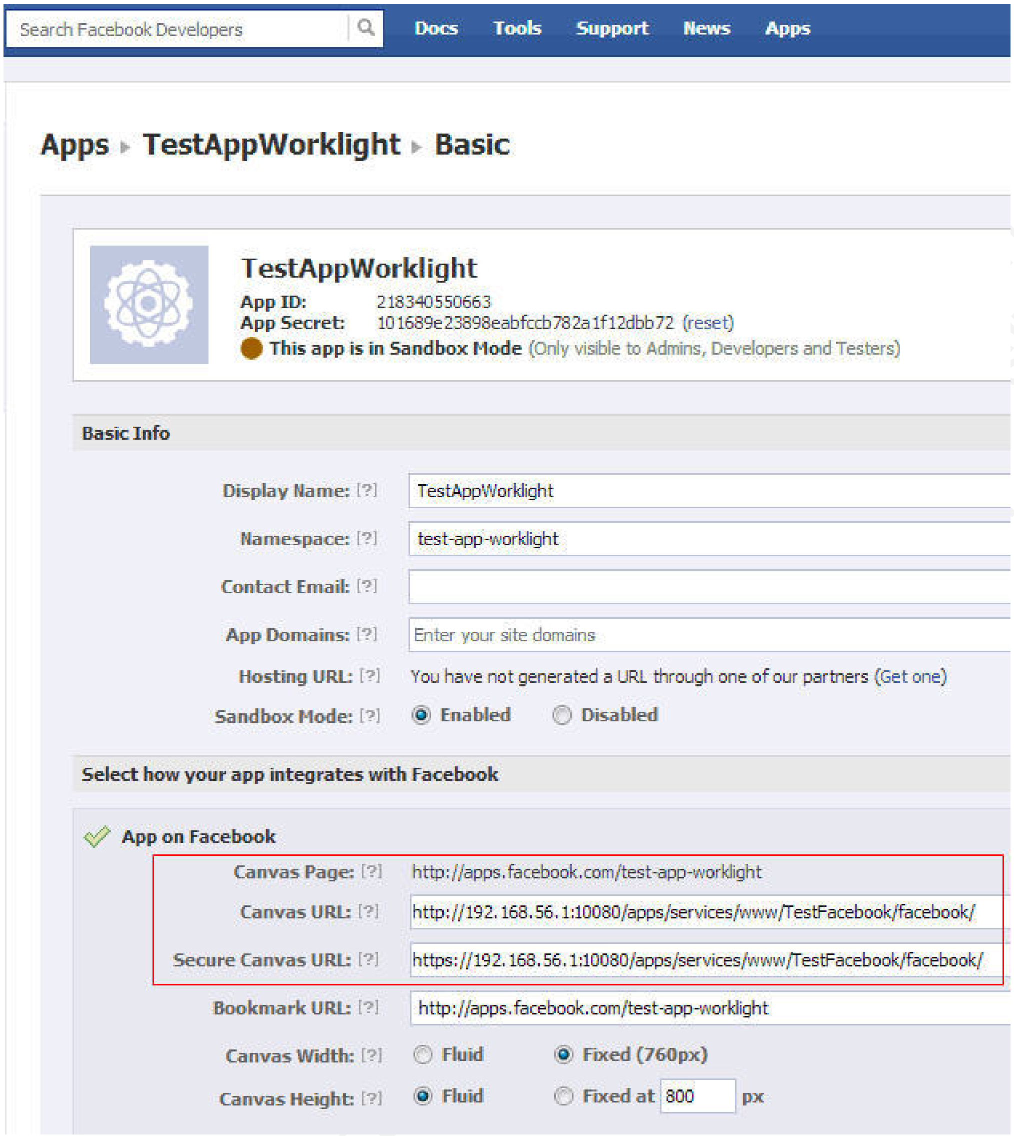
Manually migrating Facebook apps

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/`.



Apps ▸ TestAppWorklight ▸ Basic

TestAppWorklight

App ID: 218340550663

App Secret: 101689e23898eabfccb782a1f12dbb72 (reset)

This app is in Sandbox Mode (Only visible to Admins, Developers and Testers)

Basic Info

Display Name: [?] TestAppWorklight

Namespace: [?] test-app-worklight

Contact Email: [?]

App Domains: [?] Enter your site domains

Hosting URL: [?] You have not generated a URL through one of our partners (Get one)

Sandbox Mode: [?] ● Enabled ○ Disabled

Select how your app integrates with Facebook

✓ App on Facebook

Canvas Page: [?] http://apps.facebook.com/test-app-worklight

Canvas URL: [?] http://192.168.56.1:10080/apps/services/www/TestFacebook/facebook/

Secure Canvas URL: [?] https://192.168.56.1:10080/apps/services/www/TestFacebook/facebook/

Bookmark URL: [?] http://apps.facebook.com/test-app-worklight

Canvas Width: [?] ○ Fluid ● Fixed (760px)

Canvas Height: [?] ● Fluid ○ Fixed at 800 px

Figure 26. 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 or 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. 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 versions (V4.2.2, also referred to as Juno). If you already have one of these versions of Eclipse, 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://www.eclipse.org/downloads/` 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 46.

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 46.
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 “Preparing for the upgrade process” on page 271 topic can be completed before the actual installation of the new Worklight Server. The following topics, after “Starting the upgrade process” on page 276, 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 303, 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 IBM Worklight applications.

Read through 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.

Preparing 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, “Starting the upgrade process” on page 276.

Migrating 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, “Starting the upgrade process” on page 276.

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 833 stops working on those versions. The Direct Update feature works if the server-side artifacts (in this case, the `.wlapp` file) are built with the same version of Worklight Studio used to generate the mobile application. 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 840. If the application update is mandatory, another alternative is to deny access to the old application version “Remotely disabling application connectivity” on page 837.

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 46 for details.
3. Start Worklight Studio.

   See “Starting Worklight Studio” on page 48 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 711 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 711 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:

WebSphere Application Server Liberty Profile is delivered in a different way since IBM Worklight V6.0.0, and this has an impact on 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.

Since Worklight Server V6.0.0, WebSphere Application Server Liberty Core is not embedded in the IBM Installation Manager wizard of Worklight Server. Instead, it is provided as a separate IBM 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 IBM Installation Manager wizard of Worklight Server.

To prevent this existing server from being uninstalled during the upgrade process, the IBM 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 279 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.
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 "Worklight Server installation process walkthrough" on page 53.
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 282.

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 52.
2. Make sure that you have the appropriate version of IBM Installation Manager installed on the installation workstation.

   Use IBM Installation Manager V1.6.3.1 or later, especially on Windows. For more information about IBM 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 275.
4. Verify that the products that you want to update are contained in the IBM 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 IBM 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 corresponding base repository (such as Worklight Server V6.0.0) installation package from Passport Advantage or physical media.
- The appropriate V6.0.0.x installation package from FixCentral.

For a fix pack installation, you must add both repositories to the list known to IBM 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.x release (or interim fix) 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 62. This choice can influence how IBM 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.

Starting 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 “Preparing for the upgrade process” on page 271, 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.

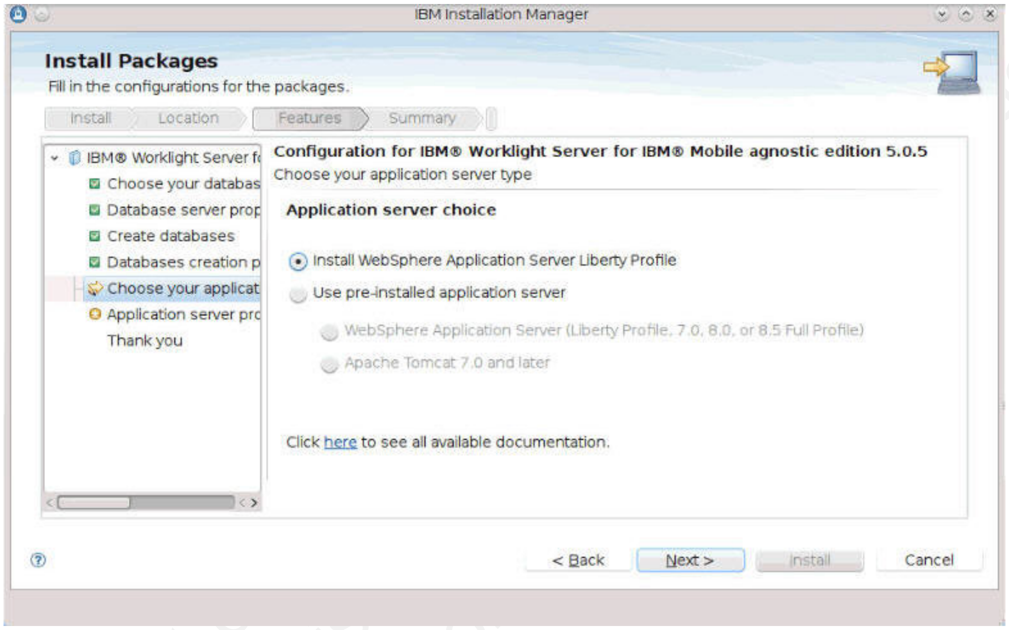
The following topics present the steps, in the order in which they must be completed:

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:



IBM Installation Manager

Install Packages

Fill in the configurations for the packages.

Configuration for IBM® Worklight Server for IBM® Mobile agnostic edition 5.0.5

Choose your application server type

Application server choice

● Install WebSphere Application Server Liberty Profile

○ Use pre-installed application server

○ WebSphere Application Server (Liberty Profile, 7.0, 8.0, or 8.5 Full Profile)

○ Apache Tomcat 7.0 and later

Click here to see all available documentation.

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/
   ```

Backing 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.

Backing 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\WAS85liberty-server\wlp`.

To back up WebSphere Application Server Full Profile:

- If your original installation was to one or more servers under the control of a deployment manager, and not a single stand-alone server:
  - Either use the WebSphere backupConfig command to backup the deployment manager node.
  - Or back up the `config` directory inside the deployment manager profile directory.
- If your original installation was to a stand-alone server:
  - Either use the WebSphere backupConfig command to backup the entire node.
  - Or back up the application server profile directory.

To back up Apache Tomcat, see its documentation 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 273.

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 IBM 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.

Verifying the ownership of files:

Before you begin the actual installation, check the ownership of all Worklight Server files.

The upcoming “Running IBM Installation Manager to perform the upgrade” on page 281 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.

Shutting 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\WAS85liberty-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.

Running 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 "Preparing for the upgrade process" on page 271 and "Starting the upgrade process" on page 276 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 64.
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 IBM Installation Manager. Then, complete the steps that are described in "Upgrading the databases and deploying the upgraded IBM Worklight project" on page 282.

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 IBM 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 273), 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 IBM 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 722.

After you upgrade the database, you must deploy the upgraded Worklight project that you create in step “Migrating your Worklight projects” on page 271. 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 748. 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 “Migrating your Worklight projects” on page 271, follow the steps that are defined in “Deploying a project WAR file and configuring the application server with Ant tasks” on page 748.

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 or schema.

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 732

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 741.

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 744.

Verifying 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>
```

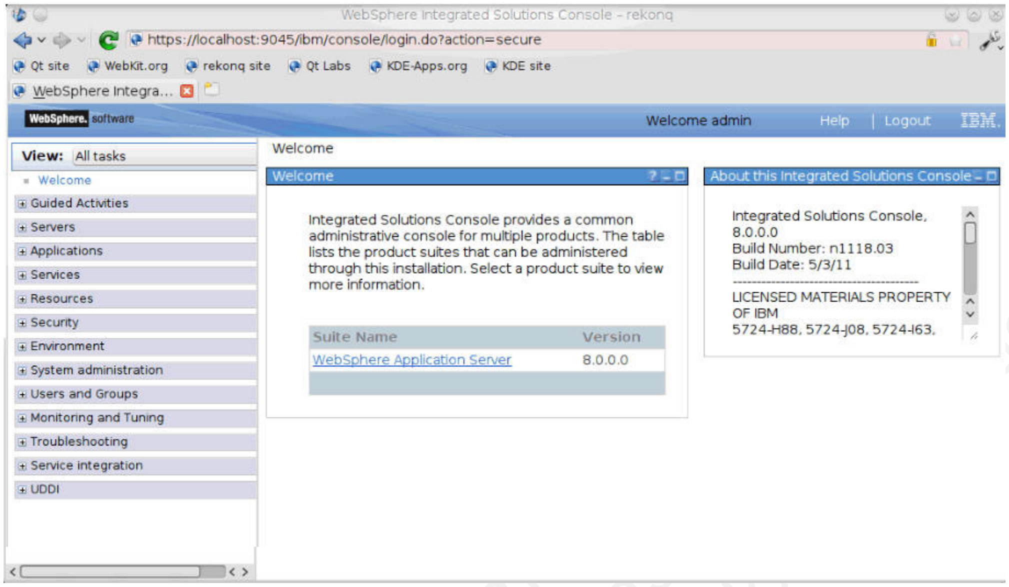
```
<!-- Declare the IBM Application Center Services application. -->
<application id="applicationcenter" name="applicationcenter" location="applicationcenter.war" type="
    <application-bnd>
        <security-role name="appcenteradmin">
            <group name="appcentergroup"/> </security-role> </application-bnd>
</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.)



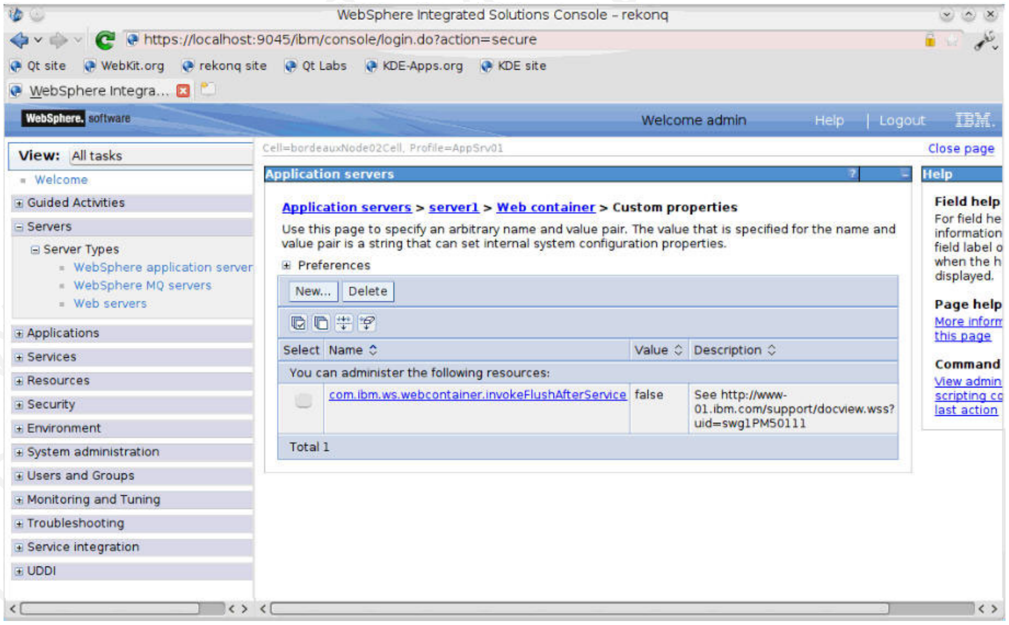
Integrated Solutions Console provides a common administrative console for multiple products. The table lists the product suites that can be administered through this installation. Select a product suite to view more information.

Suite Name	Version
WebSphere Application Server	8.0.0.0

Integrated Solutions Console, 8.0.0.0
Build Number: n1118.03
Build Date: 5/3/11

LICENSED MATERIALS PROPERTY OF IBM
5724-H88, 5724-J08, 5724-I63,

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`.



Cell=bordeauxNode02Cell, Profile=AppSrv01

Application servers > server1 > Web container > Custom properties

Use this page to specify an arbitrary name and value pair. The value that is specified for the name and value pair is a string that can set internal system configuration properties.

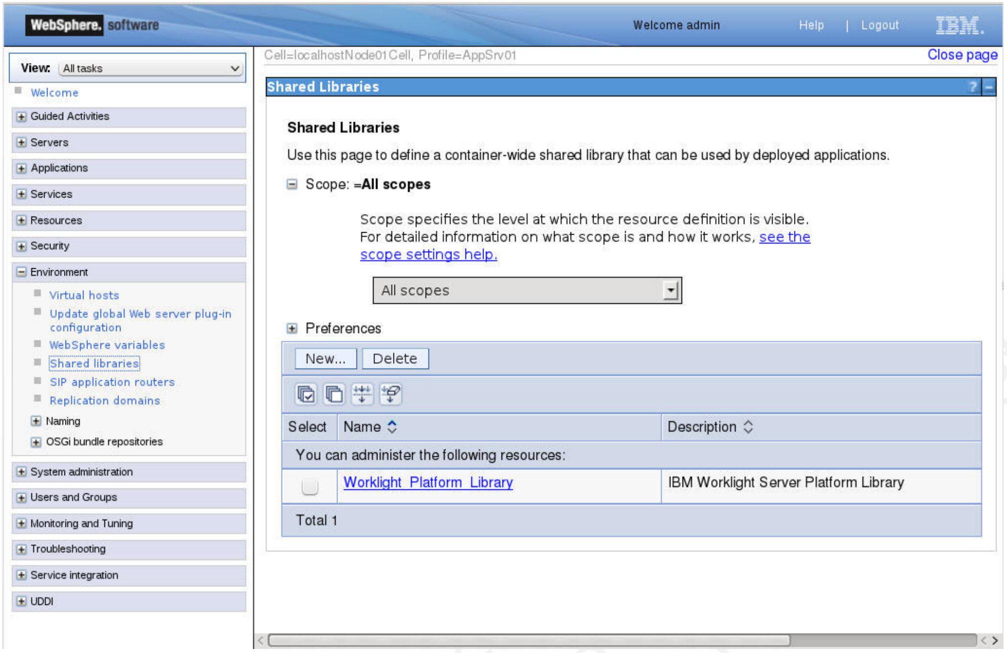
Select	Name	Value	Description
You can administer the following resources:			
□	com.ibm.ws.webcontainer.invokeFlushAfterService	false	See http://www-01.ibm.com/support/docview.wss?uid=swg1PM50111
Total 1			

Field help
For field he
information
field label o
when the h
displayed.

Page help
More inform
this page

Command
View admin
scripting co
last action

When you check the Shared Libraries, you should find one that is named `Worklight_Platform_Library`.



Cell=localhostNode01Cell, Profile=AppSrv01

Shared Libraries

Shared Libraries

Use this page to define a container-wide shared library that can be used by deployed applications.

Scope: =**All scopes**

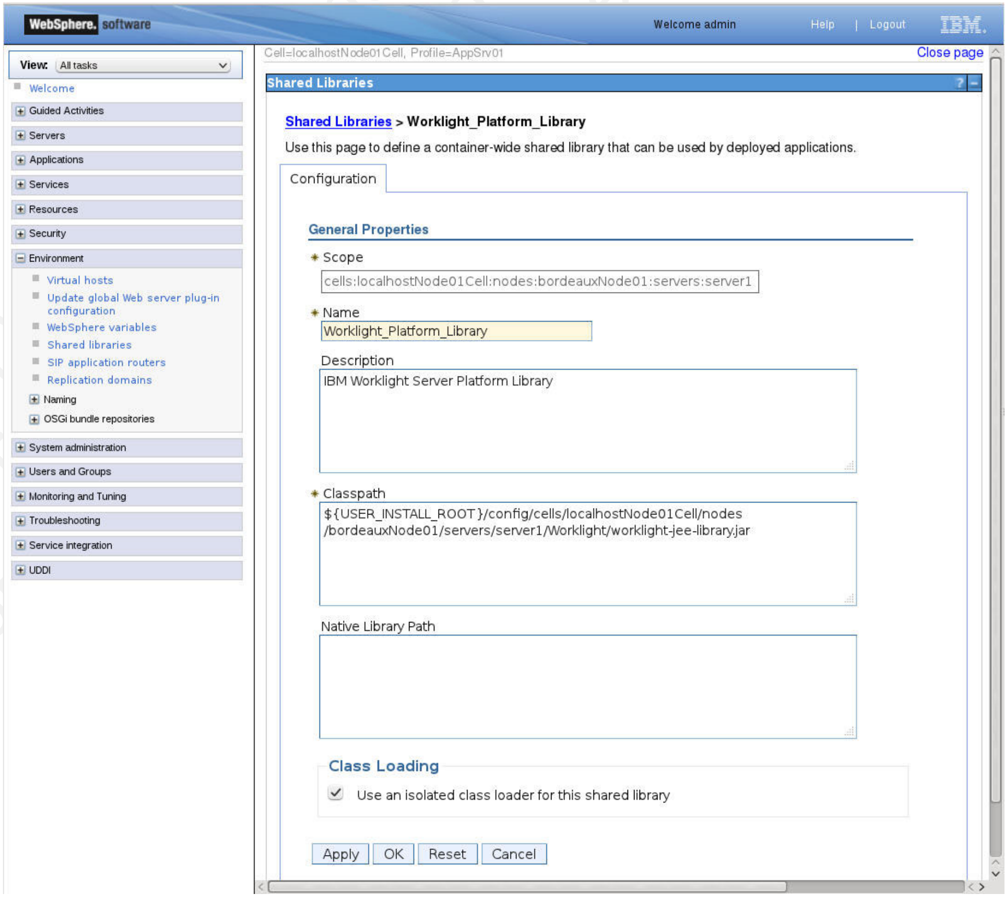
Scope specifies the level at which the resource definition is visible.
For detailed information on what scope is and how it works, see the scope settings help.

All scopes

Preferences

Select	Name	Description
You can administer the following resources:		
□	Worklight_Platform_Library	IBM Worklight Server Platform Library
Total 1		

The detail configuration of this shared library should show a classpath that consists of the `worklight-jee-library.jar`:



Cell=localhostNode01Cell, Profile=AppSrv01

Shared Libraries

Shared Libraries > Worklight_Platform_Library

Use this page to define a container-wide shared library that can be used by deployed applications.

General Properties

* Scope
cells:localhostNode01Cell:nodes:bordeauxNode01:servers:server1

* Name
Worklight_Platform_Library

Description
IBM Worklight Server Platform Library

* Classpath
${USER_INSTALL_ROOT}/config/cells/localhostNode01Cell/nodes
/bordeauxNode01/servers/server1/Worklight/worklight-jee-library.jar

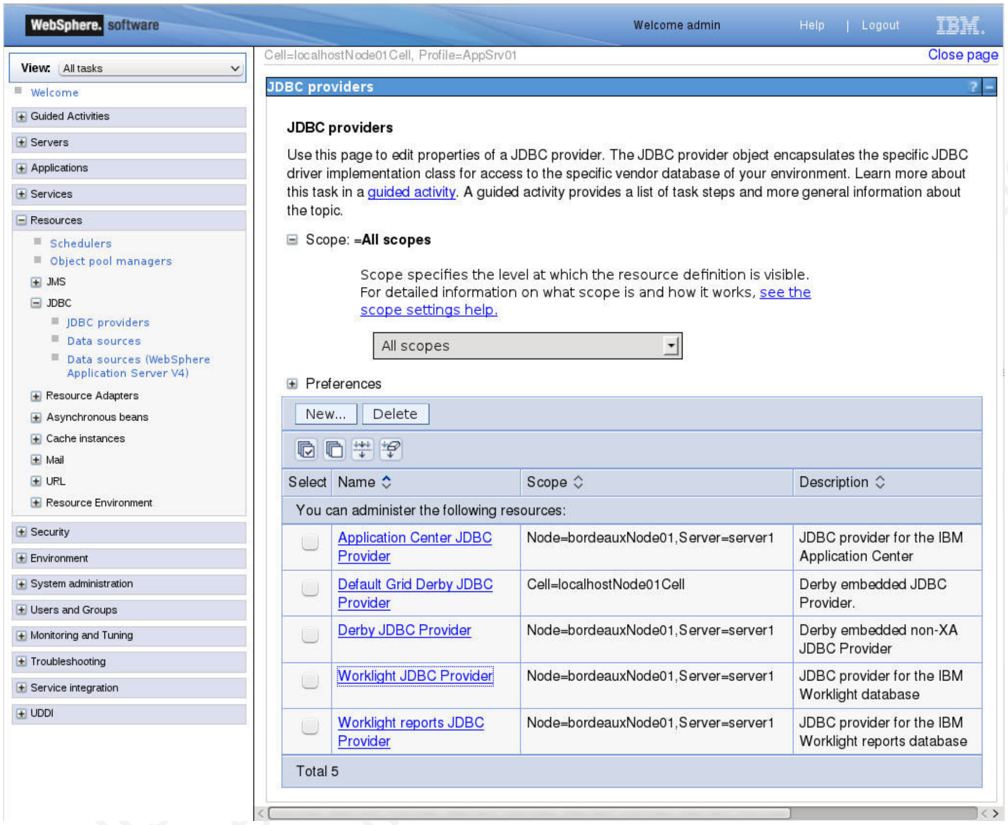
Native Library Path

Class Loading
☑ Use an isolated class loader for this shared library

When you check the JDBC providers, you should see three of them:

- `Application Center JDBC Provider`
- `Worklight JDBC provider`
- `Worklight reports JDBC Provider`

.



JDBC providers

Use this page to edit properties of a JDBC provider. The JDBC provider object encapsulates the specific JDBC driver implementation class for access to the specific vendor database of your environment. Learn more about this task in a guided activity. A guided activity provides a list of task steps and more general information about the topic.

⊟ Scope: =**All scopes**

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help.

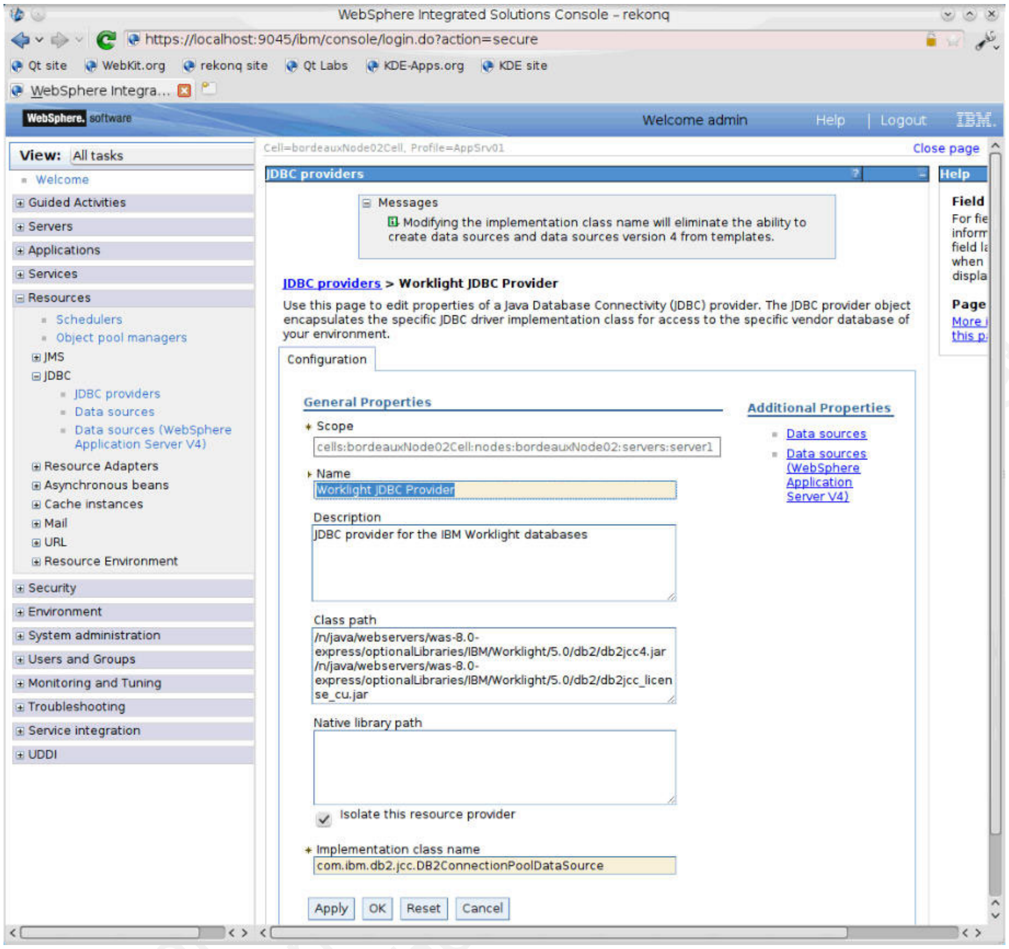
All scopes

⊞ Preferences

New... Delete

Select	Name	Scope	Description
You can administer the following resources:			
□	Application Center JDBC Provider	Node=bordeauxNode01,Server=server1	JDBC provider for the IBM Application Center
□	Default Grid Derby JDBC Provider	Cell=localhostNode01Cell	Derby embedded JDBC Provider.
□	Derby JDBC Provider	Node=bordeauxNode01,Server=server1	Derby embedded non-XA JDBC Provider
□	Worklight JDBC Provider	Node=bordeauxNode01,Server=server1	JDBC provider for the IBM Worklight database
□	Worklight reports JDBC Provider	Node=bordeauxNode01,Server=server1	JDBC provider for the IBM Worklight reports database
Total 5			

The details of the JDBC provider can look similar to the following screen capture, if the database type is DB2:



Messages

Modifying the implementation class name will eliminate the ability to create data sources and data sources version 4 from templates.

JDBC providers > Worklight JDBC Provider

Use this page to edit properties of a Java Database Connectivity (JDBC) provider. The JDBC provider object encapsulates the specific JDBC driver implementation class for access to the specific vendor database of your environment.

* Scope: cells:bordeauxNode02Cell:nodes:bordeauxNode02:servers:server1

* Name: Worklight JDBC Provider

Description: JDBC provider for the IBM Worklight databases

Class path:
/n/java/webservers/was-8.0-express/optionalLibraries/IBM/Worklight/5.0/db2/db2jcc4.jar
/n/java/webservers/was-8.0-express/optionalLibraries/IBM/Worklight/5.0/db2/db2jcc_license_cu.jar

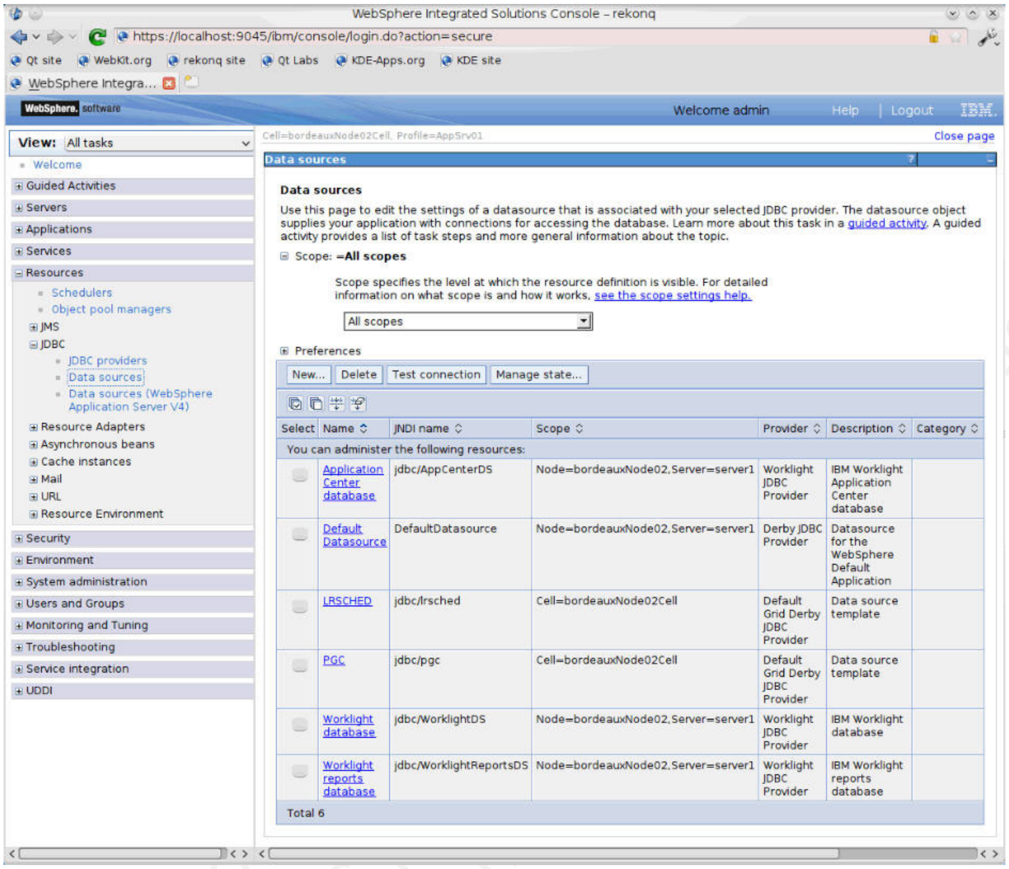
Native library path:

☑ Isolate this resource provider

* Implementation class name: com.ibm.db2.jcc.DB2ConnectionPoolDataSource

When you check the data sources, you should see three of them:

- `Worklight database`
- `Worklight reports database`
- `Application Center database`



Data sources

Use this page to edit the settings of a datasource that is associated with your selected JDBC provider. The datasource object supplies your application with connections for accessing the database. Learn more about this task in a guided activity. A guided activity provides a list of task steps and more general information about the topic.

Scope: =All scopes

Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help.

All scopes

Preferences

New... | Delete | Test connection | Manage state...

You can administer the following resources:

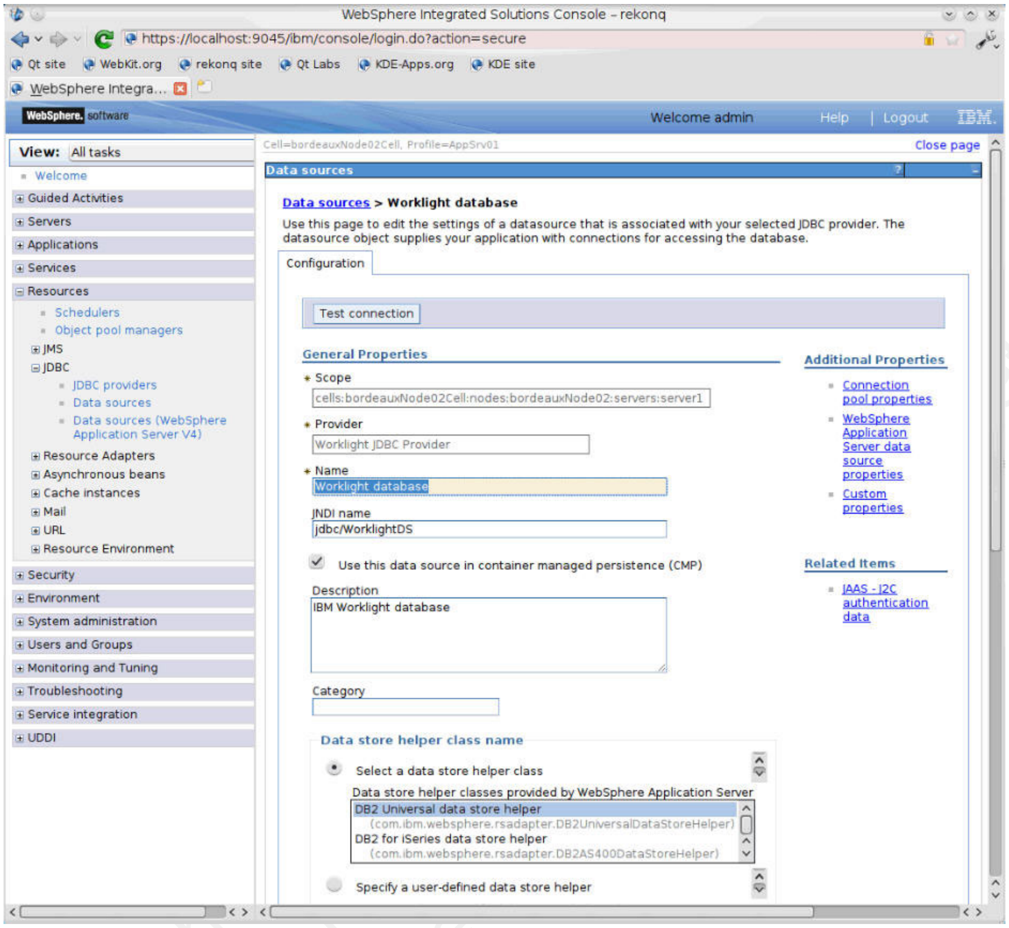
Select	Name	JNDI name	Scope	Provider	Description	Category
□	Application Center database	jdbc/AppCenterDS	Node=bordeauxNode02,Server=server1	Worklight JDBC Provider	IBM Worklight Application Center database	
□	Default Datasource	DefaultDatasource	Node=bordeauxNode02,Server=server1	Derby JDBC Provider	Datasource for the WebSphere Default Application	
□	LRSCHED	jdbc/lrsched	Cell=bordeauxNode02Cell	Default Grid Derby JDBC Provider	Data source template	
□	PGC	jdbc/pgc	Cell=bordeauxNode02Cell	Default Grid Derby JDBC Provider	Data source template	
□	Worklight database	jdbc/WorklightDS	Node=bordeauxNode02,Server=server1	Worklight JDBC Provider	IBM Worklight database	
□	Worklight reports database	jdbc/WorklightReportsDS	Node=bordeauxNode02,Server=server1	Worklight JDBC Provider	IBM Worklight reports database	

Total 6

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.



Data sources > Worklight database

Use this page to edit the settings of a datasource that is associated with your selected JDBC provider. The datasource object supplies your application with connections for accessing the database.

General Properties

* Scope: cells:bordeauxNode02Cell:nodes:bordeauxNode02:servers:server1

* Provider: Worklight JDBC Provider

* Name: Worklight database

JNDI name: jdbc/WorklightDS

☑ Use this data source in container managed persistence (CMP)

Description: IBM Worklight database

Category:

Data store helper class name

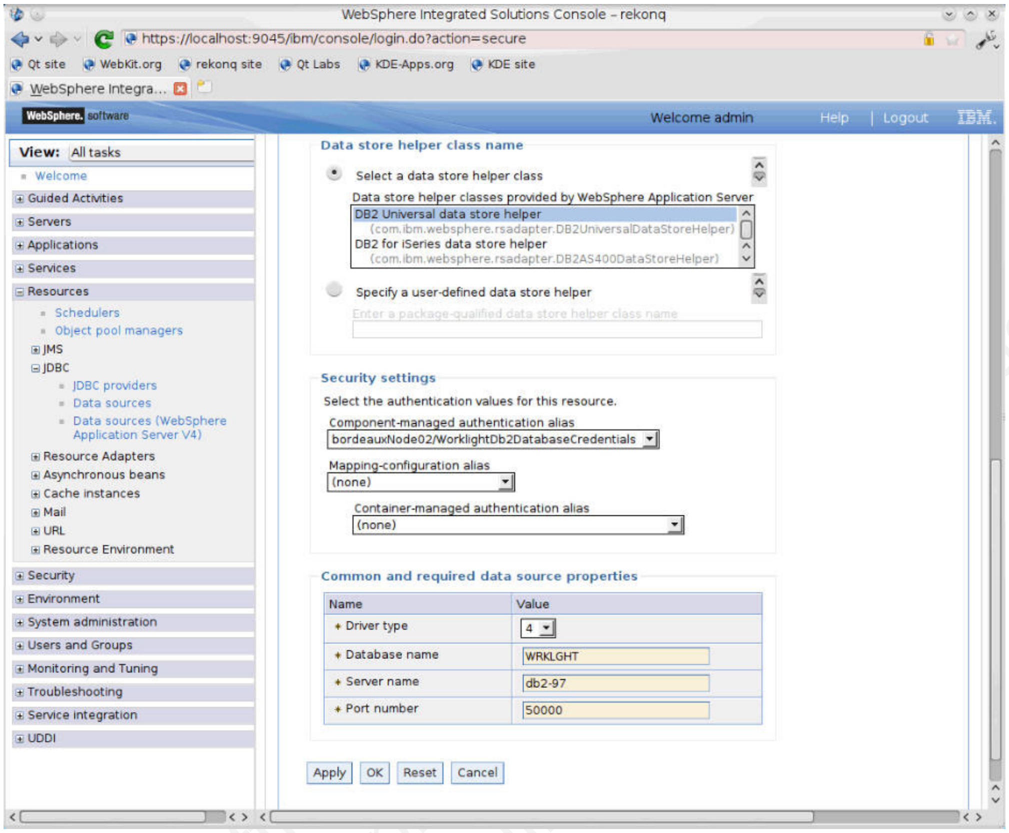
● Select a data store helper class

Data store helper classes provided by WebSphere Application Server

DB2 Universal data store helper (com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper)
DB2 for iSeries data store helper (com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper)

○ Specify a user-defined data store helper

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.



Data store helper class name

● Select a data store helper class

Data store helper classes provided by WebSphere Application Server

DB2 Universal data store helper
(com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper)
DB2 for iSeries data store helper
(com.ibm.websphere.rsadapter.DB2AS400DataStoreHelper)

○ Specify a user-defined data store helper

Enter a package-qualified data store helper class name

Security settings

Select the authentication values for this resource.

Component-managed authentication alias
bordeauxNode02/WorklightDb2DatabaseCredentials

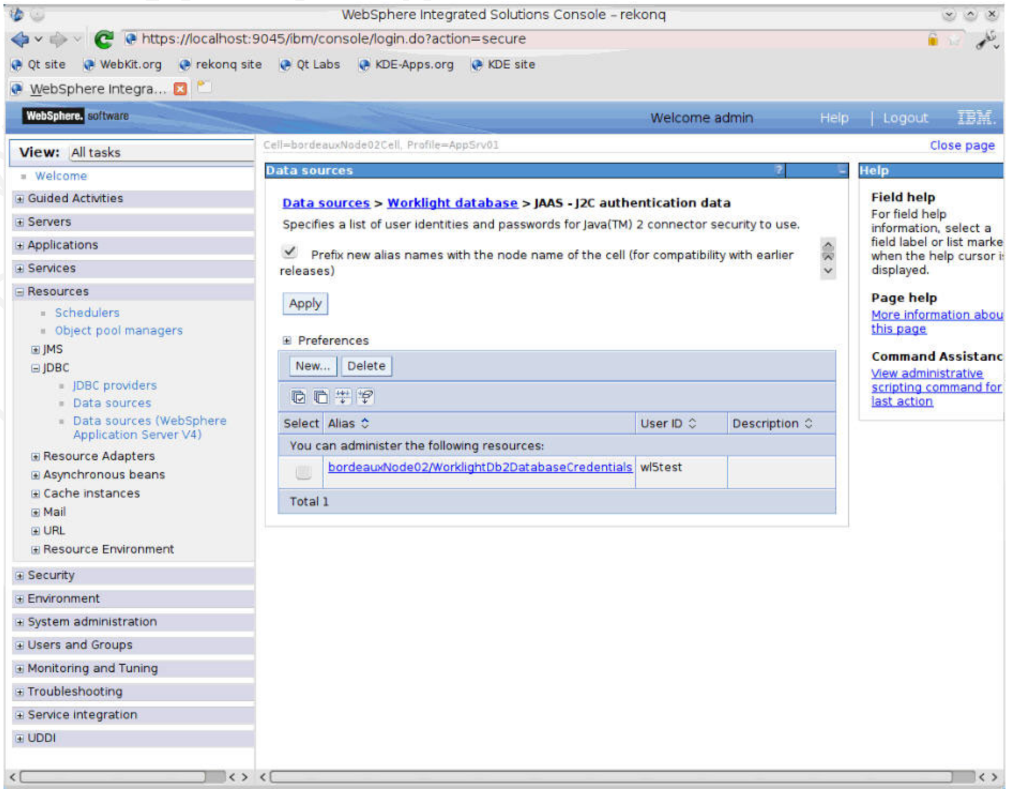
Mapping-configuration alias
(none)

Container-managed authentication alias
(none)

Common and required data source properties

Name	Value
* Driver type	4
* Database name	WRKLGHT
* Server name	db2-97
* Port number	50000

The JAAS authentication alias for DB2 can look like the following screen captures.



Cell=bordeauxNode02Cell, Profile=AppSrv01

Data sources > Worklight database > JAAS - J2C authentication data

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

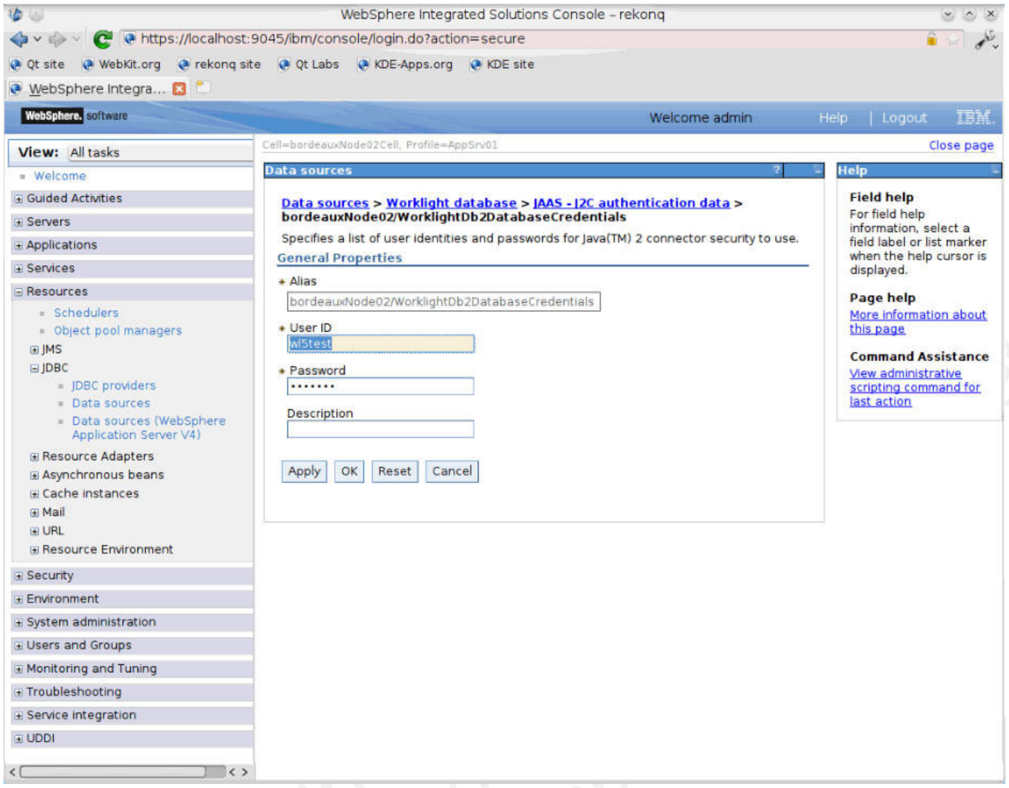
☑ Prefix new alias names with the node name of the cell (for compatibility with earlier releases)

Select	Alias	User ID	Description
You can administer the following resources:			
□	bordeauxNode02/WorklightDb2DatabaseCredentials	wl5test	
Total 1			

Field help
For field help information, select a field label or list marke when the help cursor i displayed.

Page help
More information abou this page

Command Assistanc
View administrative scripting command for last action



Data sources

Data sources > Worklight database > JAAS - J2C authentication data > bordeauxNode02/WorklightDb2DatabaseCredentials

Specifies a list of user identities and passwords for Java(TM) 2 connector security to use.

General Properties

* Alias: bordeauxNode02/WorklightDb2DatabaseCredentials

* User ID: wl5test

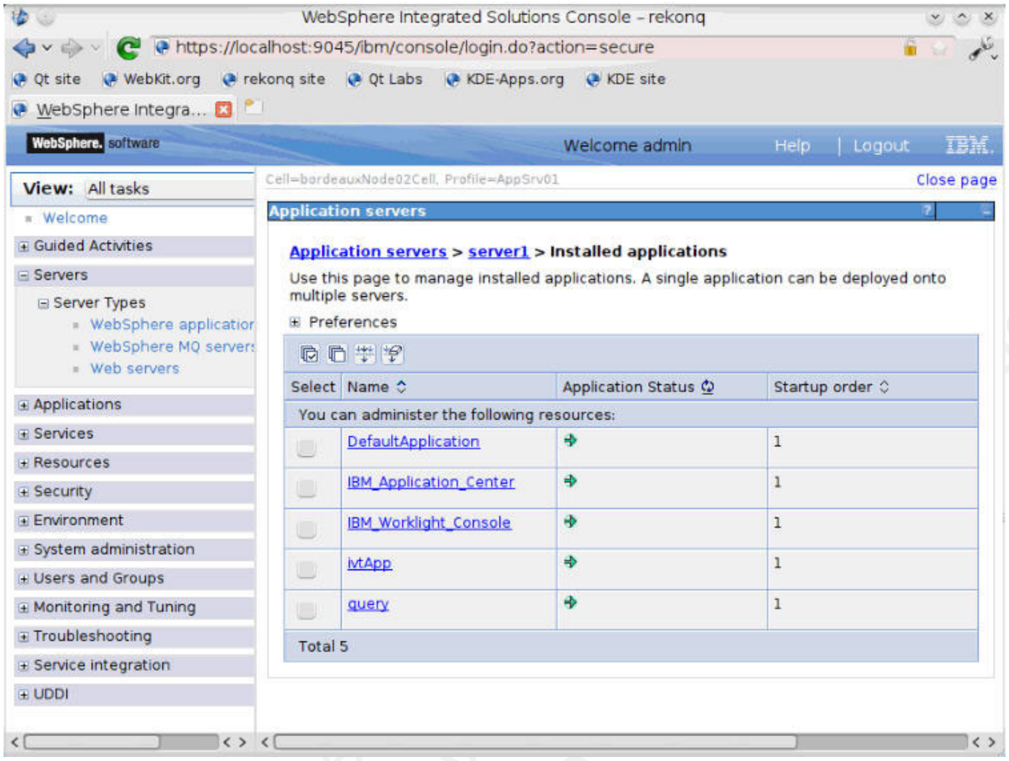
* Password: •••••••

Description:

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).

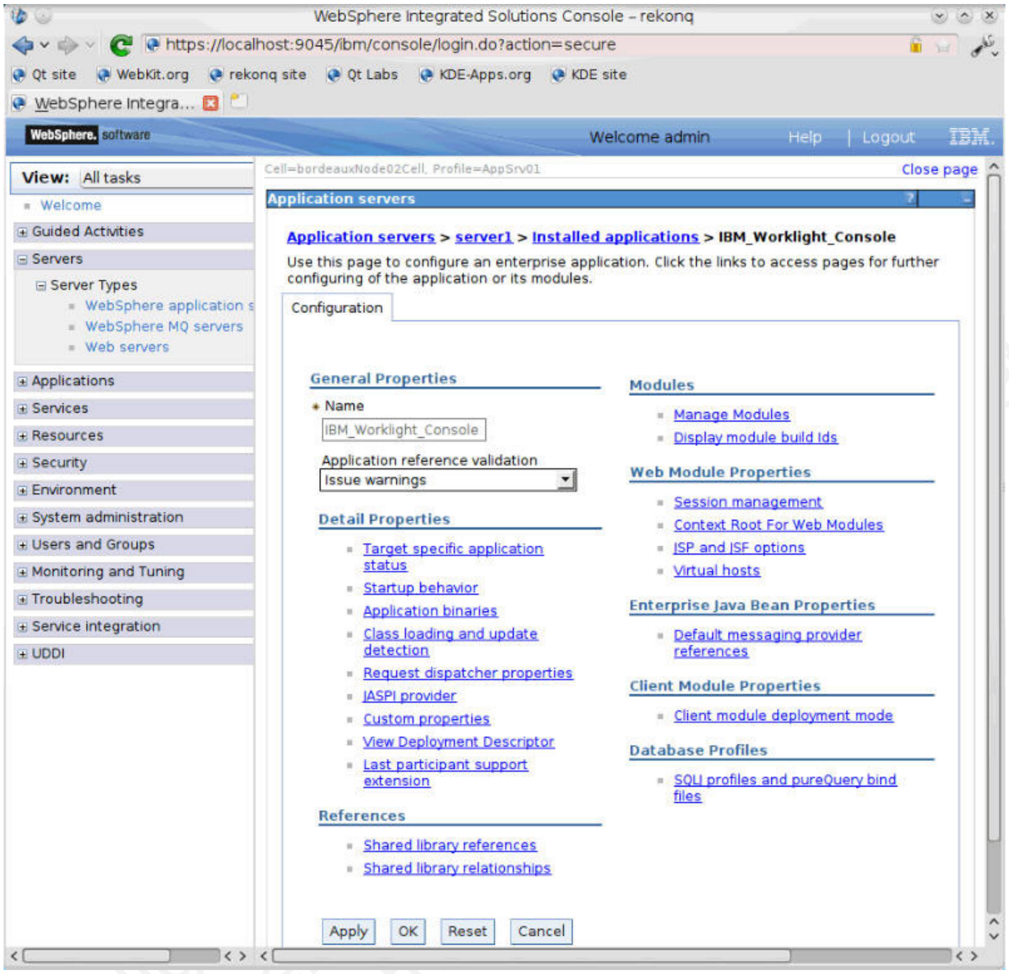


Application servers > server1 > Installed applications

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

Select	Name	Application Status	Startup order
You can administer the following resources:			
□	DefaultApplication	➜	1
□	IBM_Application_Center	➜	1
□	IBM_Worklight_Console	➜	1
□	ivtApp	➜	1
□	query	➜	1
Total 5			

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.

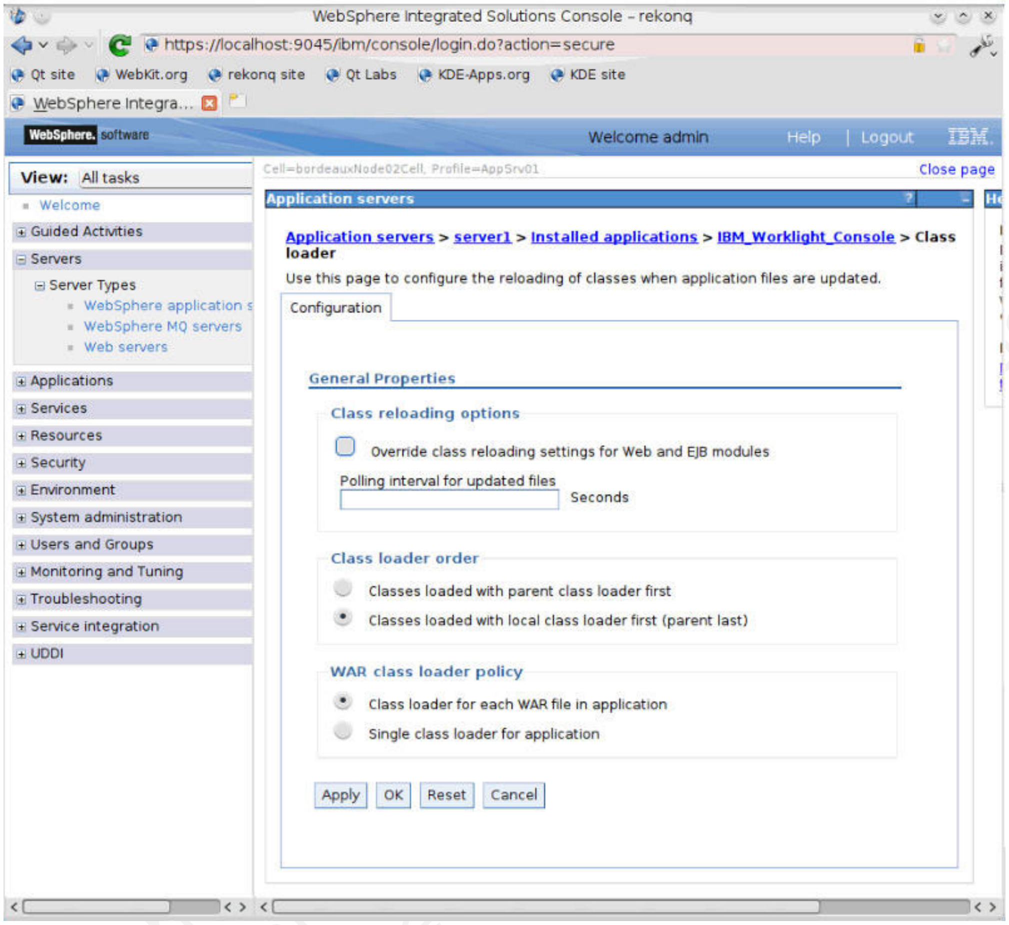


Application servers > server1 > Installed applications > IBM_Worklight_Console

Use this page to configure an enterprise application. Click the links to access pages for further configuring of the application or its modules.

* Name
IBM_Worklight_Console

Application reference validation
Issue warnings



Application servers > server1 > Installed applications > IBM_Worklight_Console > Class loader

Use this page to configure the reloading of classes when application files are updated.

Configuration

General Properties

Class reloading options

□ Override class reloading settings for Web and EJB modules

Polling interval for updated files Seconds

Class loader order

○ Classes loaded with parent class loader first

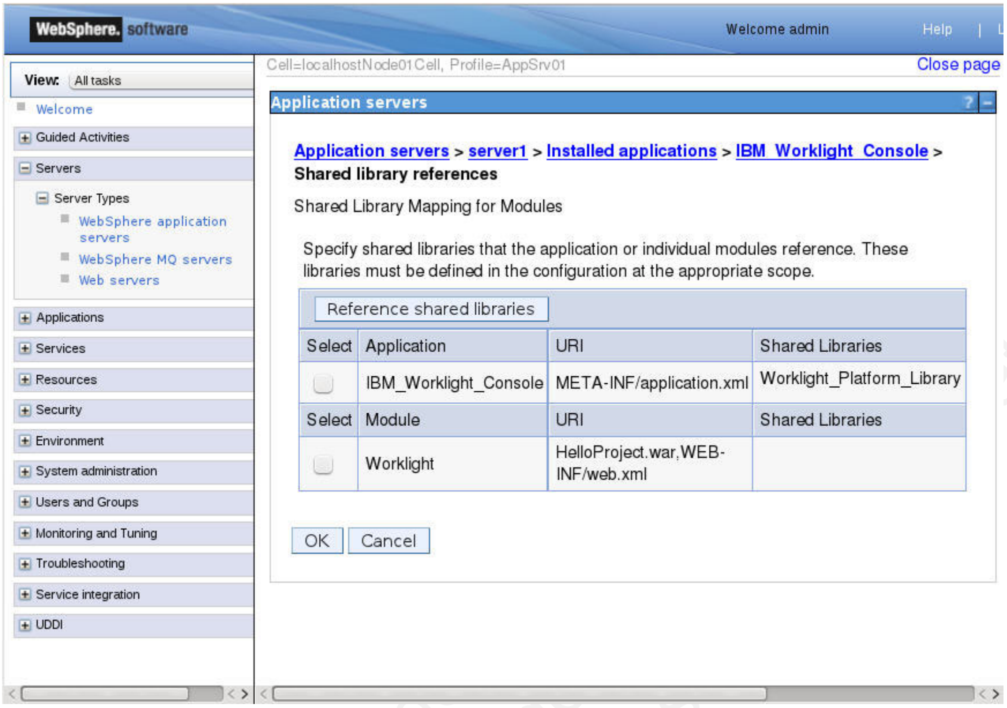
● Classes loaded with local class loader first (parent last)

WAR class loader policy

● Class loader for each WAR file in application

○ Single class loader for application

When you look at the shared library references of the application IBM_Worklight_Console, you should see a reference to the Worklight_Platform_Library.



Cell=localhostNode01Cell, Profile=AppSrv01

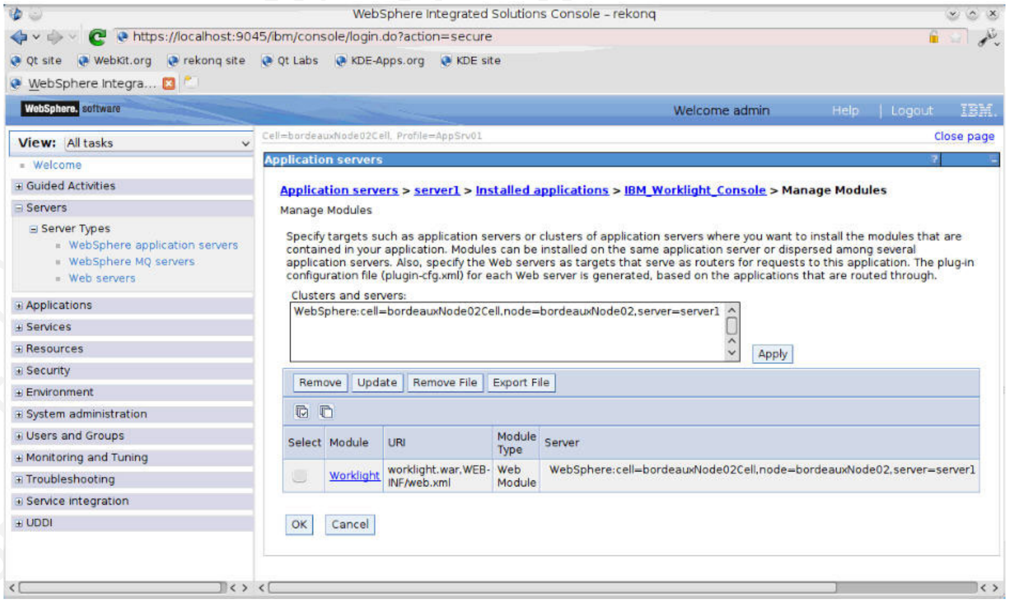
Application servers > server1 > Installed applications > IBM_Worklight_Console > **Shared library references**

Shared Library Mapping for Modules

Specify shared libraries that the application or individual modules reference. These libraries must be defined in the configuration at the appropriate scope.

Select	Application	URI	Shared Libraries
□	IBM_Worklight_Console	META-INF/application.xml	Worklight_Platform_Library
Select	**Module**	**URI**	**Shared Libraries**
□	Worklight	HelloProject.war,WEB-INF/web.xml	

The IBM_Worklight_Console application has a single module, Worklight, corresponding to the project WAR file.



Cell=bordeauxNode02Cell, Profile=AppSrv01

Application servers > server1 > Installed applications > IBM_Worklight_Console > **Manage Modules**

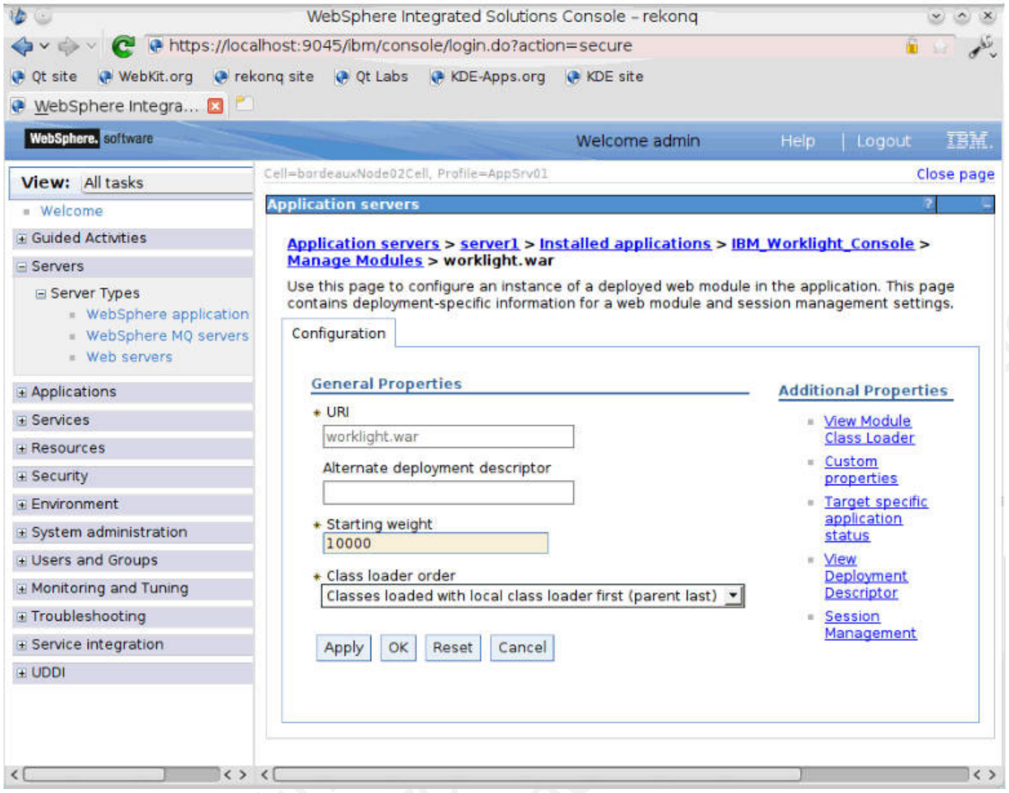
Manage Modules

Specify targets such as application servers or clusters of application servers where you want to install the modules that are contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed through.

Clusters and servers:
WebSphere:cell=bordeauxNode02Cell,node=bordeauxNode02,server=server1

Select	Module	URI	Module Type	Server
□	Worklight	worklight.war,WEB-INF/web.xml	Web Module	WebSphere:cell=bordeauxNode02Cell,node=bordeauxNode02,server=server1

The module settings for each module among the Worklight Server applications should contain a class loader order of parent last.



Cell=bordeauxNode02Cell, Profile=AppSrv01

Application servers > server1 > Installed applications > IBM_Worklight_Console > Manage Modules > worklight.war

Use this page to configure an instance of a deployed web module in the application. This page contains deployment-specific information for a web module and session management settings.

General Properties

* URI: worklight.war

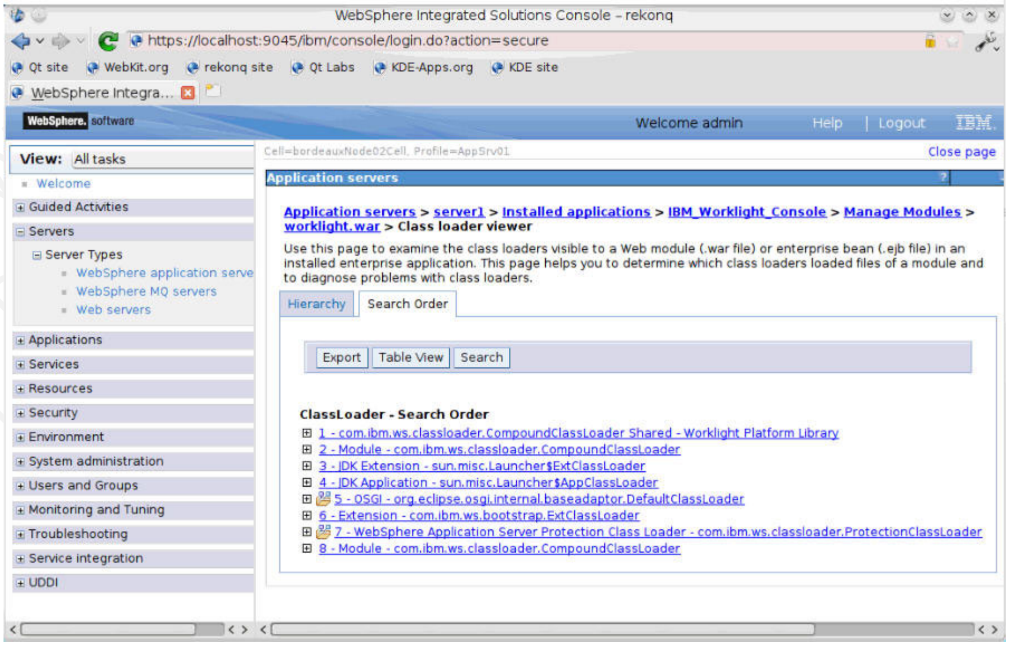
Alternate deployment descriptor:

* Starting weight: 10000

* Class loader order: Classes loaded with local class loader first (parent last)

Additional Properties

The class loader viewer for module worklight.war in the IBM Worklight Console should mention the Worklight_Platform_Library.



Cell=bordeauxNode02Cell, Profile=AppSrv01

Application servers > server1 > Installed applications > IBM_Worklight_Console > Manage Modules > worklight.war > Class loader viewer

Use this page to examine the class loaders visible to a Web module (.war file) or enterprise bean (.ejb file) in an installed enterprise application. This page helps you to determine which class loaders loaded files of a module and to diagnose problems with class loaders.

ClassLoader - Search Order

⊞ 1 - com.ibm.ws.classloader.CompoundClassLoader Shared - Worklight Platform Library
⊞ 2 - Module - com.ibm.ws.classloader.CompoundClassLoader
⊞ 3 - JDK Extension - sun.misc.Launcher$ExtClassLoader
⊞ 4 - JDK Application - sun.misc.Launcher$AppClassLoader
⊞ 5 - OSGI - org.eclipse.osgi.internal.baseadaptor.DefaultClassLoader
⊞ 6 - Extension - com.ibm.ws.bootstrap.ExtClassLoader
⊞ 7 - WebSphere Application Server Protection Class Loader - com.ibm.ws.classloader.ProtectionClassLoader
⊞ 8 - Module - com.ibm.ws.classloader.CompoundClassLoader

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/realm-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
```

Verifying and updating 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 180
- "Setting up IBM Worklight in an IBM WebSphere Application Server Liberty Profile farm" on page 191

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>
```

Configuring the Application Center

Set configuration options for the Worklight Application Center.

Since 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 138 and in:

- For WebSphere Application Server Liberty Profile: "Configuring WebSphere Application Server Liberty Profile" on page 140
- For WebSphere Application Server Full Profile: "Configuring WebSphere Application Server full profile" on page 139
- For Apache Tomcat: "Configuring Apache Tomcat" on page 141

The endpoint URI should be configured as documented in "Defining the endpoint of the application resources" on page 165 and in:

- For WebSphere Application Server Liberty Profile: "Configuring the endpoint of the application resources (Liberty profile)" on page 167
- For WebSphere Application Server Full Profile: "Configuring the endpoint of the application resources (Full Profile)" on page 166
- For Apache Tomcat: "Configuring the endpoint of the application resources (Apache Tomcat)" on page 168

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.

Starting the application server:

Restart the Worklight Application Center.

If all previous tasks are completed, now you can start the application server.

Finishing the migration of 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 .wlapp file and adapters) that you create in step "Migrating your Worklight projects" on page 271.

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 833) stops working on those versions. For instructions about how to re-enable direct update for those applications, see "Migrating your Worklight projects" on page 271.

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 IBM Installation Manager.
2. Install the old version of Worklight Server with IBM Installation Manager, specifying the same installation parameters that you used previously.
3. Restore the databases. For more information, see "Backing up the databases" on page 278.
4. Restore the application server. For more information, see "Backing up the application server" on page 278.
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 308.

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 270.

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 52 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`, `.wlapp`, 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 307.

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 307.

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 467.

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 453.

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 700 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 27.

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 the `addItem` method, as defined in the WL.OptionsMenu class.

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 `createIdenity` method.

Changes in sessions configuration

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

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

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

Changes in Application Center

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

Dojo iOS fixes

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

About this task

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

Procedure

1. Open your workspace using IBM Worklight Studio.
2. Right-click the project you want to migrate and click **Properties**.
3. Click **Project Facets**.
4. Uncheck **Web 2.0-> Dojo Toolkit** to uninstall the tooling for Dojo. It does not uninstall Dojo.
5. Click **OK** to close the project **Properties** page.
6. Find the folder ***{PROJECT_NAME}*** > **dojo** and delete all the children under this folder. Do not delete the ***{PROJECT_NAME}*** > **dojo** folder. It must be empty.
7. Right-click the project again and go to the **Properties** page again.
8. Click **Project Facets**.
9. Check **Web 2.0-> Dojo Toolkit** to install the tooling for Dojo. You have now upgraded to a new Dojo version.
10. You might have to upgrade some application content to work with Dojo. Run and test your application to make sure things work with this version.

Dojo 1.8.3 code migration

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

About this task

Make the following changes to migrate your existing code.

Procedure

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

Example

The following code snippet shows the new technique:

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

                                });
                        });
}
```

Migrating from 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 414.

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 138 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 307.

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.

Chapter 8. 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 apps. 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.

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. `.wlapp` 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 the project WAR file" on page 714.

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 and files:

- 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.
- A `legal` folder, for you to store all the license-related documents.
- An `application-descriptor.xml` file that contains the application metadata. For more information about this file, see "The application descriptor" on page 331.

- A `build-settings.xml` file, for you to prepare minification and concatenation configurations for each environment. For more information about this file, see "IBM Worklight application build settings" on page 516.

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, and Windows 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
`Hybrid Application`	To create a Worklight project with an initial hybrid application **Note:** You can choose to use IBM Worklight Application Framework (Beta code) to help you rapidly create your application, including the UI, and manage communication with back-end services. For more information about how to use this framework, see "Developing hybrid applications with IBM Worklight Application Framework" on page 425.
`Inner Application`	To create a Worklight project with an initial inner application and point to a built shell component
`Shell Component`	To create a Worklight project with an initial shell component application
`Native Application`	To create a Worklight project with an initial Native application

4. Optional: Click **Configure Framework Library** to set your IBM Worklight Application Framework library preferences.

5. Optional: Click **Configure JavaScript Libraries**, and select any of the following options to add the corresponding support to the application:

Option	Description
Add jQuery Mobile	To add jQuery Mobile support to the application. You must identify the directory where the required files for jQuery Mobile are located.
Add Sencha Touch	To add Sencha Touch support to the application. You must identify the directory where the required files for Sencha Touch are located.
Add Dojo Toolkit	To add the Dojo facet and Dojo support to the application. When you build a mobile web application, Dojo is included to create the native application, such as an iPhone or Android application. **Note:** If you chose to use IBM Worklight Application Framework for your application, **Dojo Mobile** is automatically selected. **Dojo Mobile** cannot be cleared because Dojo Mobile is required for every application created with IBM Worklight Application Framework.

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. Optional: If you want to use IBM Worklight Application Framework (Beta code) for your application, select **Use Worklight Application Framework (beta)**. To know more about how to use IBM Worklight Application Framework , see “Developing hybrid applications with IBM Worklight Application Framework” on page 425.

d. Optional: To change the settings for the IBM Worklight Application Framework library, click **Configure Framework Library**.

e. Optional: To add JavaScript libraries to your application, click **Configure JavaScript Libraries**, and 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 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`.

   Note: If you selected **Use Worklight Application Framework (beta)** in step c, **Dojo Mobile** is automatically selected. **Dojo Mobile** cannot be cleared because the Dojo Mobile is required for every app created with IBM Worklight Application Framework.

- 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.

Results

An application of the type that you selected is now visible in your IBM Worklight project, and the application descriptor opens.

Note: In case you chose to use IBM Worklight Application Framework for your new hybrid application, the IBM Worklight Application Framework editor opens, instead of the application descriptor.

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 317.
- Create an IBM Worklight project, and your application in it as its first application, as described in “Creating IBM Worklight projects” on page 316

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 315
- 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 338

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” on page 320

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 322 and “Developing native applications” on page 467.

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 696.

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 699.

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 699.

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 700.

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
|
+---Java Resources
+---JavaScript Resources
+---adapters
+---apps
|   \---Application Name
|       |   application-descriptor.xml
|       |   build-settings.xml
|       |
|       +---android
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |   |   |   Application Name.iml
|       |   |   |   AndroidManifest.xml
|       |   |   |   project.properties
|       |   |   |
|       |   |   +---assets
|       |   |   |   |   wlclient.properties
|       |   |   |   |
|       |   |   |   +---featurelibs
|       |   |   |   +---www (*)
|       |   |   +---gen (*)
|       |   |   +---libs
|       |   |   +---res
|       |   |   +---src
|       |   +---nativeResources
|      |
|       +---blackberry
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |       |   config.xml
|       |       |   icon.png
|       |       |   splash.png
|       |       |   .wldata
|       |       |
|       |       +---ext
|       |       |       WLExtension.jar
|       |       |
|       |       +---www (*)
|       |
|       +---blackberry10
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---nativeResources
|       |   |   |
|       |   |   +---www (*)
|       +---common
|       |       |   index.html
|       |   |
|       |   +---css
|       |   +---images
|       |   +---js
|       |
|       +---ipad
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |   |   |   buildtime.sh
|       |   |   |   config.xml
|       |   |   |   Entitlements-Debug.plist
|       |   |   |   Entitlements-Release.plist
|       |   |   |   main.m
|       |   |   |   Project Name Application Name-Info.plist
|       |   |   |   Project Name Application Name_Prefix.pch
|       |   |   |   README.txt
|       |   |   |   worklight.plist
```

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 323.

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.

Table 51. An IBM 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`	

Table 51. An IBM Worklight project has the following structure: (continued)

<project-name>			**Root project folder**
		conf	Worklight Server configuration files, such as worklight.properties and authenticationConfig.xml
		java	Java code that must be compiled and packaged into jar files deployable to the Worklight Server
		lib	Pre-compiled libraries that must be deployed to the Worklight Server
	services		Description, at development stage, of back-end services discovered for consumption by the applications in the project

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 356.

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.

Table 52. 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
		`index.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
	`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`		
	`build-settings.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 as follows:
  - 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\www`. 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 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. Rebuild the application by clicking **Run As** > **Run on Worklight Development Server** or **Run As** > **Build...**.

    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. Rebuild the application by clicking **Run As** > **Run on Worklight Development Server** or **Run As** > **Build...**.

    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\www`, **OR**
    2. Add the new (replacement) image to `blackberry10\nativeResources\www`.
    3. Rebuild the application by clicking **Run As** > **Run on Worklight Development Server** or **Run As** > **Build...**.

    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. Rebuild the application by clicking **Run As** > **Run on Worklight Development Server** or **Run As** > **Build...**.

- 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. Rebuild the application by clicking **Run As** > **Run on Worklight Development Server** or **Run As** > **Build...**.

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

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 53. 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`	

Table 53. Application icons (continued)

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

Table 53. 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 can change as long as it is listed in the `plist` file.	An icon that is displayed on the device springboard. Size depends on iOS version and target device. iOS6.1 and earlier: • Non-retina display: 57 by 57 pixels • Retina display: 114 by 114 pixels iOS7: • 120 by 120 pixels	`iphone\native\resources`	`\iphone\nativeResources\Resources`
Windows Phone 8	`Background.png` `ApplicationIcon.png`	Both icons are used to identify the application. `Background.png` is displayed on the device home screen, and must be 300 by 300 pixels. `ApplicationIcon.png` is displayed in the list of applications, and must be 100 by 100 pixels.	`windowsphone8\native`	

Table 53. Application icons (continued)

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

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

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="6.1.0.00.20140126-0630">
xmlns="http://www.example.com/application-descriptor" xmlns:xsi=http://www.w3.org/2001/XMLSchema-ins
xsi:schemaLocation="http://www.example.com/application-descriptor
../../../../../gadgets/application-descriptor/src/main/resources/schema/application-descriptor.xsd">
```

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

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

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

```
<displayName>First 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 2014 </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>index.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>
<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 />
<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** (except for web apps). 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

```
<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 335 for details of this element.

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

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

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

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

The app user can use the IBM Worklight settings screen to change the address of the 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 335 for details of this element.

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

The <windowsPhone8> element has three subelements:

- The <uuid> subelement is used to uniquely identify a Windows Phone 8 application on the device. It is automatically generated by the 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 “Setting up push notifications for Windows Phone 8” on page 590.
- 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.

```
<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 466 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 464 for more details.
- Use the `<certificate>` element to sign the AIR application before you publish it. See "Signing Adobe AIR applications" on page 465 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 454

The <features> element

Since 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.
      ...
    <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 507.

The <cacheManifest> element

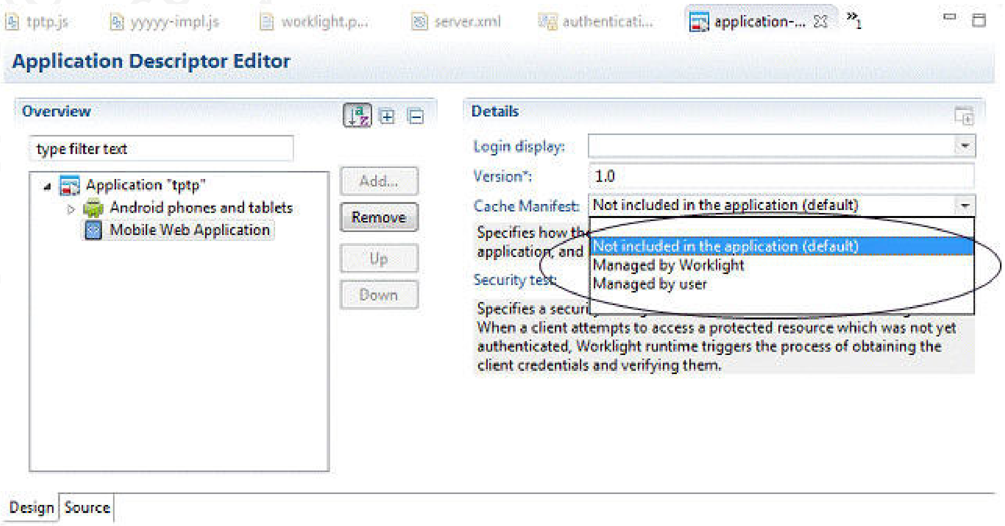
A new element now exists in the Application Descriptor (application-descriptor.xml) named <cacheManifest>. Using this element, you can manage and edit the contents of the application cache for Desktop Browser and Mobile Web applications. and thus control which resources are fetched when the application 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 54. `cacheManifest` properties

Property	Description
`generated`	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: • For Desktop Browser environments – all resources are under NETWORK, which means: no cache at all. • For Mobile Web environments – all resources are under CACHE, which means: cache everything. In **`generated`** 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.
`no-use`	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.
`user`	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.

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:



Login display:

Version*: 1.0

Cache Manifest: Not included in the application (default)

Not included in the application (default)
Managed by Worklight
Managed by user

Specifies how th application, and

Security test:

Specifies a securi

When a client attempts to access a protected resource which was not yet authenticated, Worklight runtime triggers the process of obtaining the client credentials and verifying them.

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 platformVersion="6.1.0.00.20140126-0630" id="MyProj" xmlns="http://www.worklight.com/app
      ...
    <mobileWebApp cacheManifest="generated"/>
    <desktopBrowser cacheManifest="generated"/>
</application>
```

For more information about the Cache Manifest, see “Application cache management in Desktop Browser and Mobile Web apps” on page 511.

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>`) was 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 27.

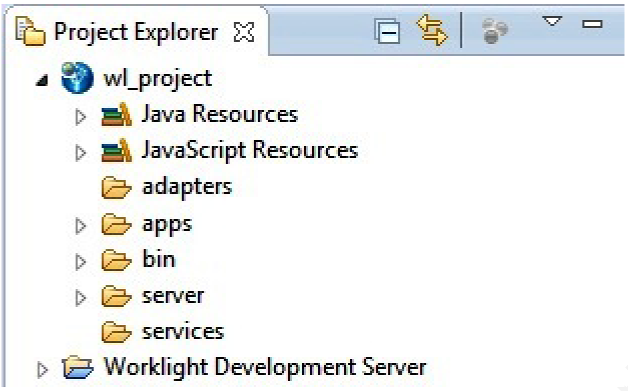
Setting up a new IBM Worklight environment for your application

With Worklight Studio, you can build applications for different mobile, desktop, or web environments within your IBM Worklight project.

Before you begin

Note: Starting IBM Worklight V6.1.0, the structure of Project Explorer is simplified and focuses on three main components that the user is interested in: adapters, apps, and services. The following figure shows the directory structure.

Figure 28. Project Explorer showing simplified structure



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 has 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 316. To learn how to create an application, see “Creating an application in an IBM Worklight project” on page 317

   You can see your new application within your IBM Worklight project in the Project Explorer.

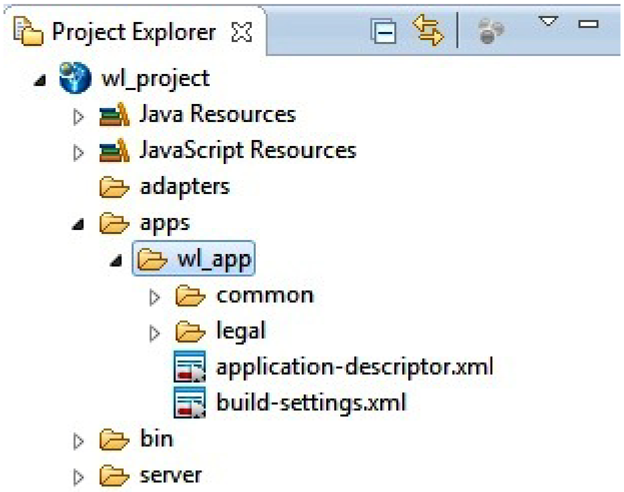
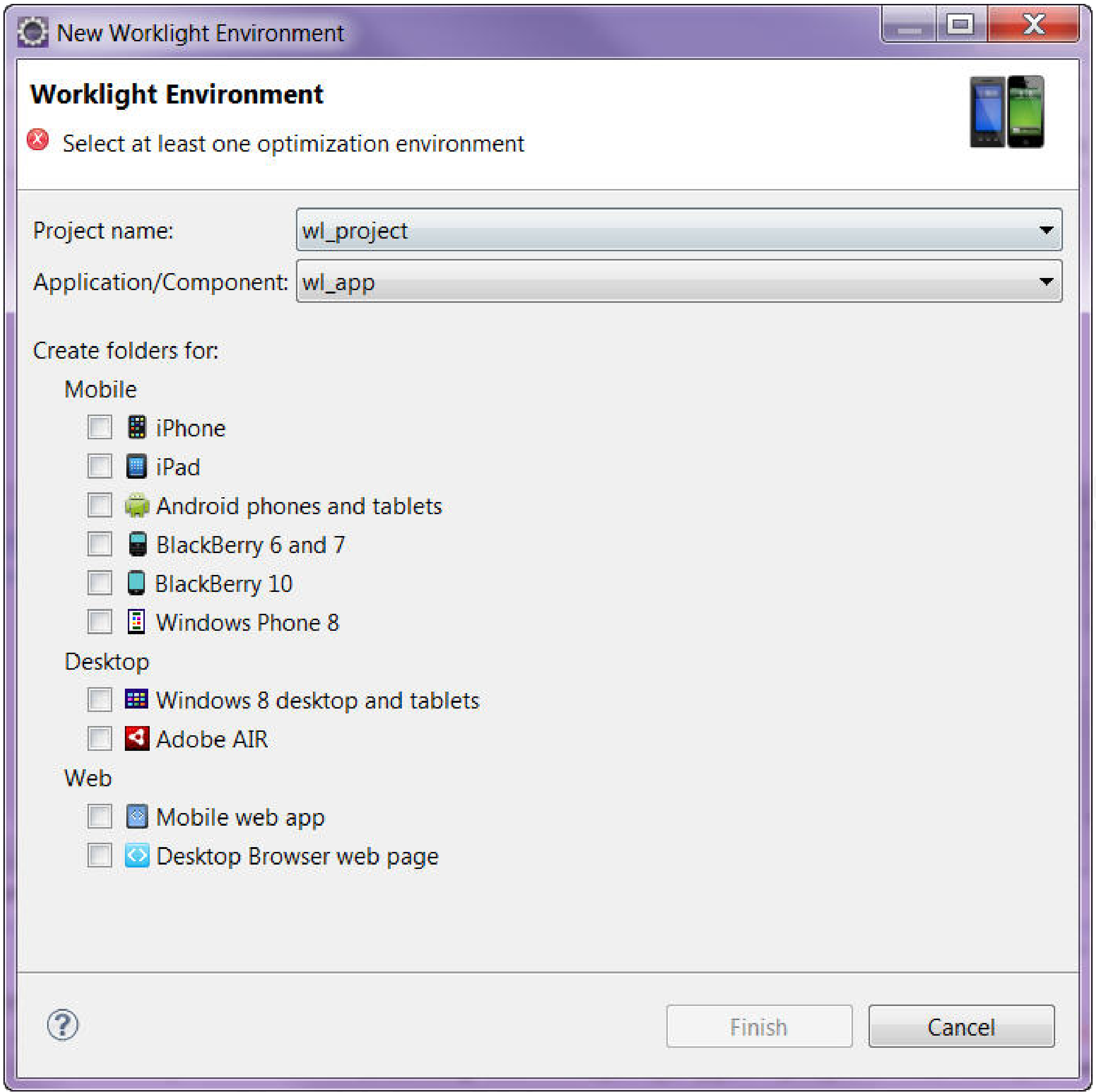


Figure 29. Your IBM Worklight application folder

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.



New Worklight Environment

Worklight Environment

Select at least one optimization environment

Project name: wl_project

Application/Component: wl_app

Create folders for:

Mobile

- □ iPhone
- □ iPad
- □ Android phones and tablets
- □ BlackBerry 6 and 7
- □ BlackBerry 10
- □ Windows Phone 8

Desktop

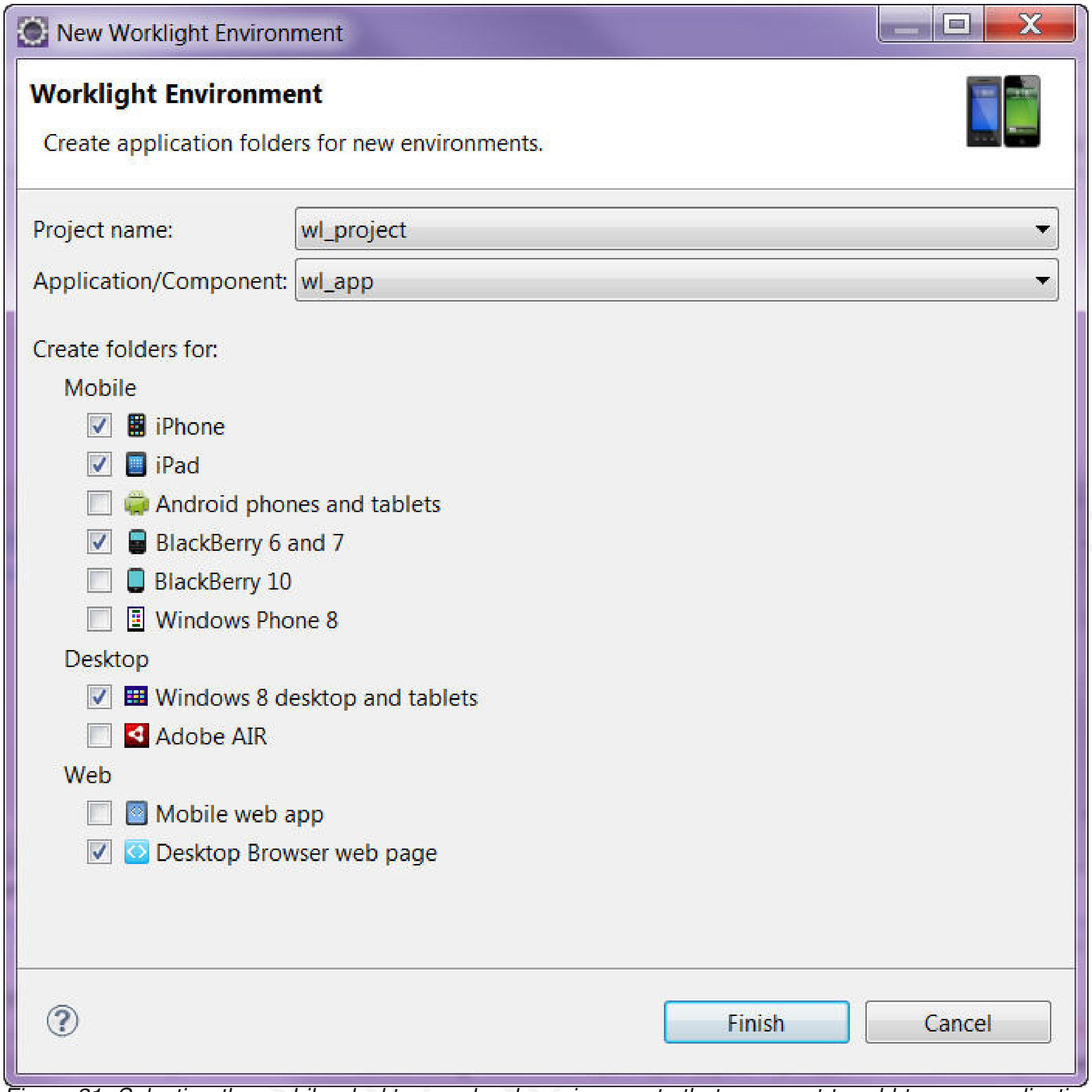
- □ Windows 8 desktop and tablets
- □ Adobe AIR

Web

- □ Mobile web app
- □ Desktop Browser web page

Figure 30. 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.



New Worklight Environment

Worklight Environment

Create application folders for new environments.

Project name: wl_project

Application/Component: wl_app

Create folders for:

Mobile

- ☑ iPhone
- ☑ iPad
- ☐ Android phones and tablets
- ☑ BlackBerry 6 and 7
- ☐ BlackBerry 10
- ☐ Windows Phone 8

Desktop

- ☑ Windows 8 desktop and tablets
- ☐ Adobe AIR

Web

- ☐ Mobile web app
- ☑ Desktop Browser web page

Figure 31. 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.

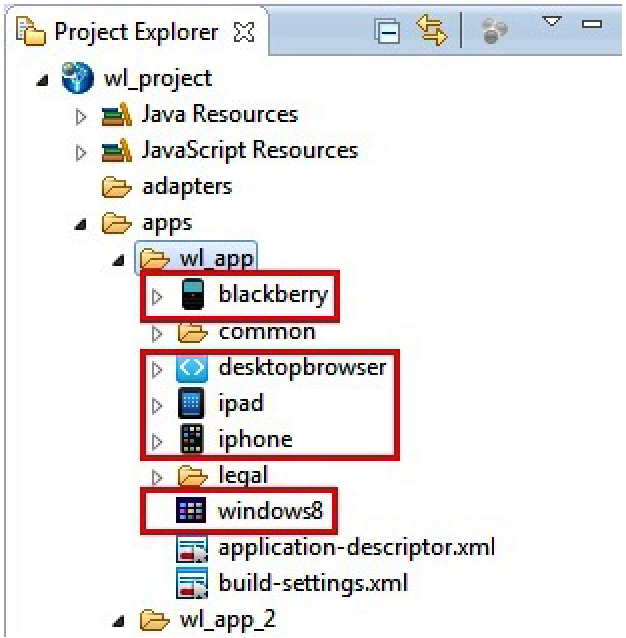


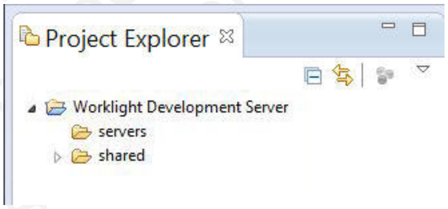
Figure 32. IBM Worklight application folder that contains folders for the environments you selected

The Worklight Development Server and the Worklight Console

Information about the Worklight Development Server, how it is viewed in the Worklight Console, and how to access it for Java remote debugging.

Since IBM Worklight V6.0.0, the Jetty server was 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 **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 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.

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 *Changing the Worklight Server associated with a project* procedure described in “Working with multiple Worklight Servers in Worklight Studio” on page 345.

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.

Since 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
```

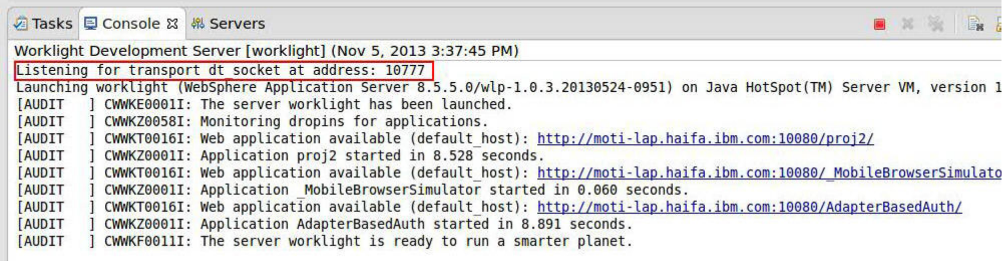
The default `<port>` after Worklight Studio installation is now `10080`. So the Worklight Console URL for a project named **myProject** becomes:

```
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 displays the console for the server instance for which the context root was set using the **Run As** > **Build Settings and Deploy Target** 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.

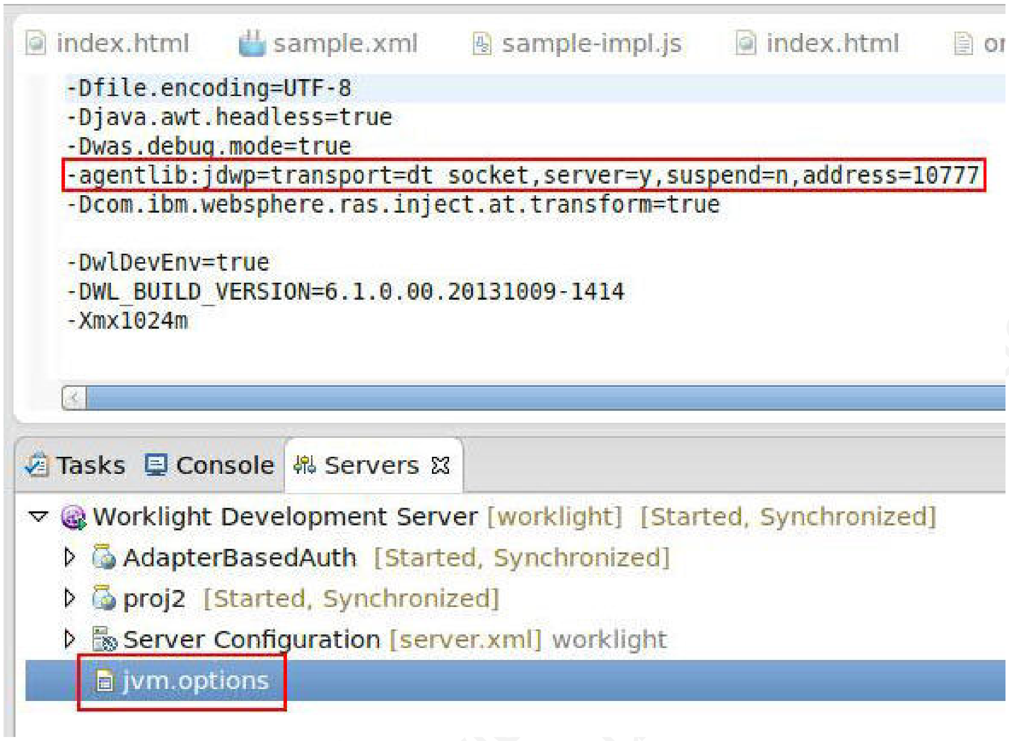
Java remote debug and the Worklight Development Server

Since IBM Worklight V6.1.0, the Liberty profile instance used as the Worklight Development Server has Java remote debug enabled. The default port is 10777, and can be viewed in the Console view of Worklight Studio when the server is started:



```
Worklight Development Server [worklight] (Nov 5, 2013 3:37:45 PM)
Listening for transport dt_socket at address: 10777
Launching worklight (WebSphere Application Server 8.5.5.0/wlp-1.0.3.20130524-0951) on Java HotSpot(TM) Server VM, version 1
[AUDIT   ] CWWKE0001I: The server worklight has been launched.
[AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://moti-lap.haifa.ibm.com:10080/proj2/
[AUDIT   ] CWWKZ0001I: Application proj2 started in 8.528 seconds.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://moti-lap.haifa.ibm.com:10080/_MobileBrowserSimulato
[AUDIT   ] CWWKZ0001I: Application _MobileBrowserSimulator started in 0.060 seconds.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://moti-lap.haifa.ibm.com:10080/AdapterBasedAuth/
[AUDIT   ] CWWKZ0001I: Application AdapterBasedAuth started in 8.891 seconds.
[AUDIT   ] CWWKF0011I: The server worklight is ready to run a smarter planet.
```

This default port can be changed by editing the `jvm.options` file in the Worklight Studio Servers view:



```
-Dfile.encoding=UTF-8
-Djava.awt.headless=true
-Dwas.debug.mode=true
-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=10777
-Dcom.ibm.websphere.ras.inject.at.transform=true

-DwlDevEnv=true
-DWL_BUILD_VERSION=6.1.0.00.20131009-1414
-Xmx1024m
```

```
Worklight Development Server [worklight] [Started, Synchronized]
  AdapterBasedAuth [Started, Synchronized]
  proj2 [Started, Synchronized]
  Server Configuration [server.xml] worklight
   jvm.options
```

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 343, 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 **Run As** > **Run on Worklight Development Server**.

Path to the Worklight Development Server and its console

As previously noted, 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 936.

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.

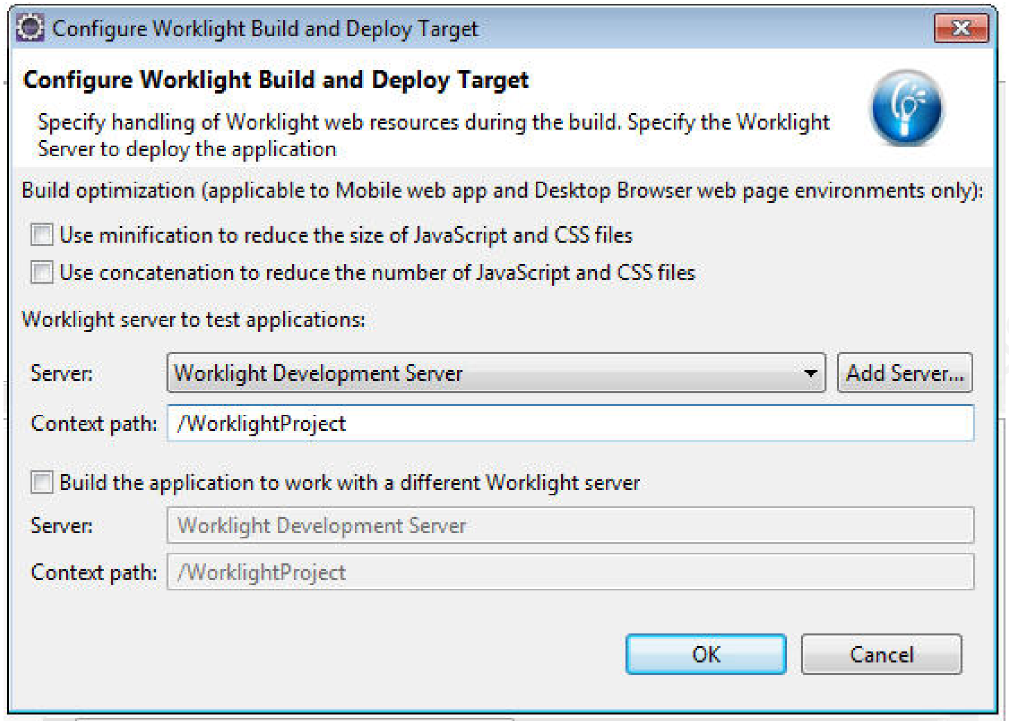
Every change that is made to the project source that is related to the project WAR (under the `/server` folder) is automatically built and deployed to the current target server. The database connector JAR files and Worklight JAR file are also automatically deployed to this target server when you deploy 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 target test server or change the Worklight project *context root* (which Worklight Server it is associated with) by right-clicking the application and selecting **Run As** > **Build Settings and Deploy Target**.

This action displays the following window:



Configure Worklight Build and Deploy Target

Configure Worklight Build and Deploy Target

Specify handling of Worklight web resources during the build. Specify the Worklight Server to deploy the application

Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

☐ Use minification to reduce the size of JavaScript and CSS files

☐ Use concatenation to reduce the number of JavaScript and CSS files

Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

☐ Build the application to work with a different Worklight server

Server: Worklight Development Server

Context path: /WorklightProject

If the Worklight Server instance you want to associate with this project is visible in the **Server** drop-down list, select it, update the **Context path** if necessary, and click **OK**. The outcome of this action is:

1. The project's WAR file is automatically updated with the new context root value the next time you build.
2. After rebuilding and deploying the application, the new context root is also saved in the client-side files.

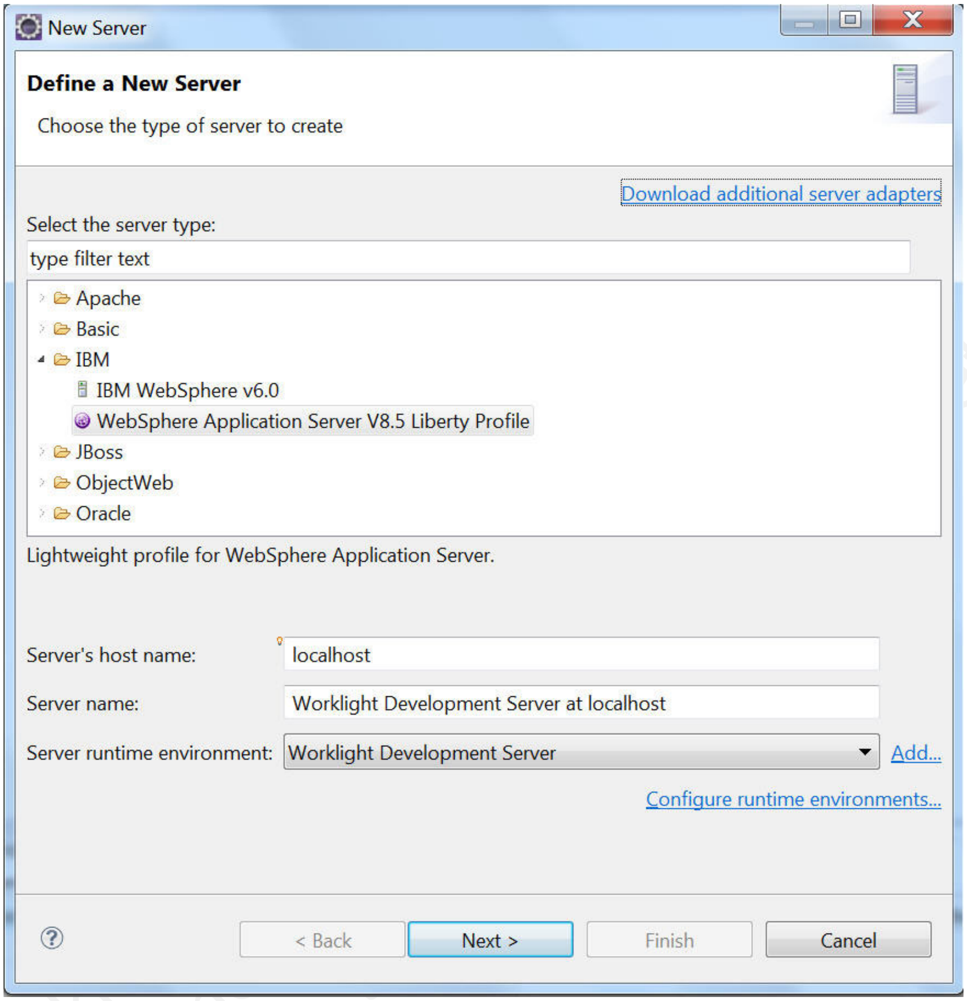
The selected server now becomes your default test t Worklight Server. This action also changes the URL under the **Open Worklight Console** menu command, so that it now points to the new server.

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 Worklight Server

If the Worklight Server instance you want to select is not visible in the **Server** dropdown list, you can add a new Worklight Server 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 Configure Worklight Build and Deploy Target window, click **Add Server** to display the following window:


New Server
Define a New Server
Choose the type of server to create
Download additional server adapters
Select the server type:
type filter text
Apache
Basic
IBM
IBM WebSphere v6.0
WebSphere Application Server V8.5 Liberty Profile
JBoss
ObjectWeb
Oracle
Add...
Configure runtime environments...
< Back
Next >
Finish
Cancel

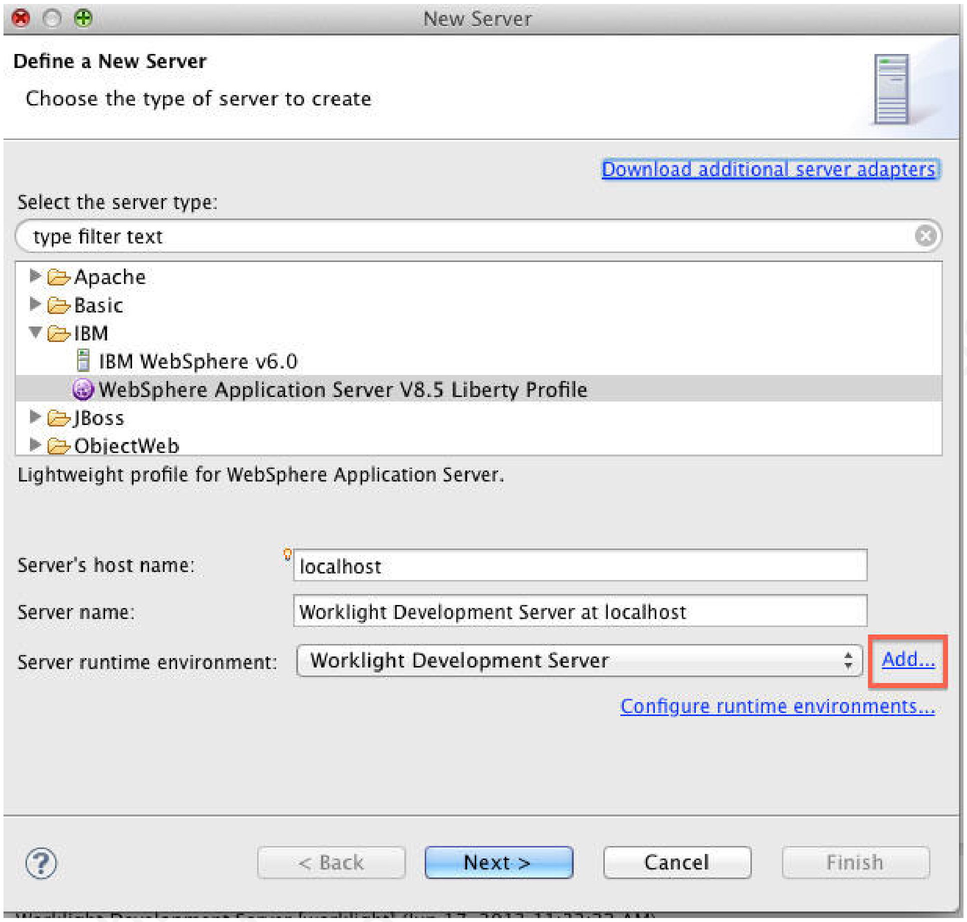
Lightweight profile for WebSphere Application Server.

Server's host name: localhost

Server name: Worklight Development Server at localhost

Server runtime environment: Worklight Development Server

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):



New Server

Define a New Server

Choose the type of server to create

Download additional server adapters

Select the server type:

type filter text

- Apache
- Basic
- IBM
  - IBM WebSphere v6.0
  - WebSphere Application Server V8.5 Liberty Profile
- JBoss
- ObjectWeb

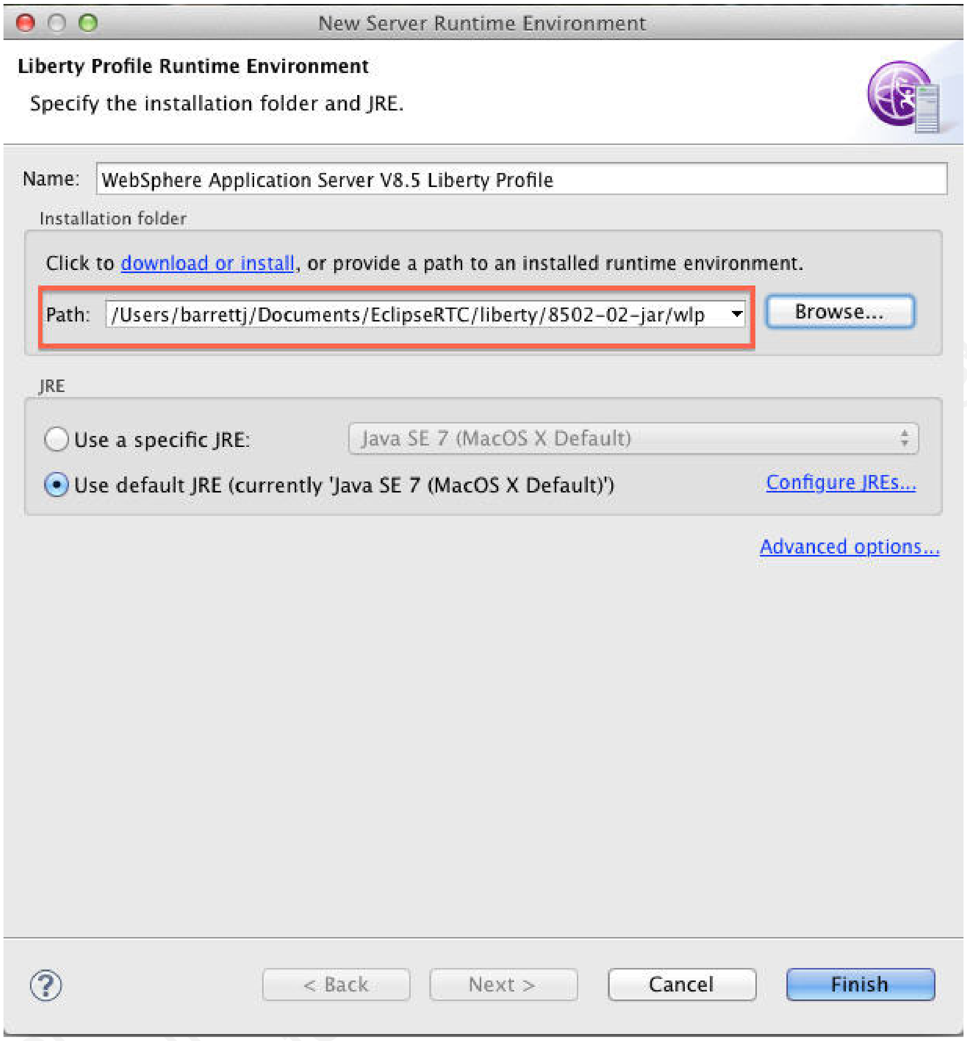
Lightweight profile for WebSphere Application Server.

Server's host name: localhost

Server name: Worklight Development Server at localhost

Server runtime environment: Worklight Development Server

3. On the resulting screen, set **Path** to point to the directory containing the new external Liberty profile server.



New Server Runtime Environment

Liberty Profile Runtime Environment

Specify the installation folder and JRE.

Name: WebSphere Application Server V8.5 Liberty Profile

Installation folder

Click to download or install, or provide a path to an installed runtime environment.

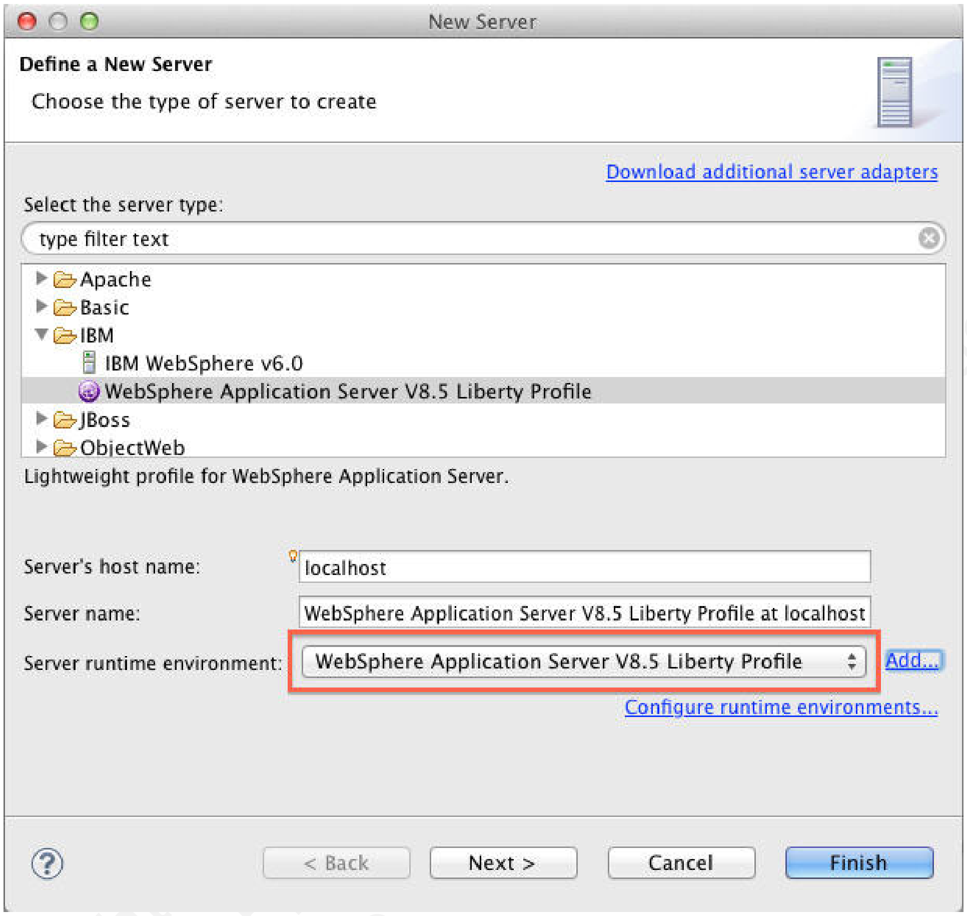
Path: /Users/barrettj/Documents/EclipseRTC/liberty/8502-02-jar/wlp

JRE

○ Use a specific JRE: Java SE 7 (MacOS X Default)

● Use default JRE (currently 'Java SE 7 (MacOS X Default)')

4. After you add the new server, it displays under Server runtime environment.



New Server

Define a New Server

Choose the type of server to create

Select the server type:

Server's host name: localhost

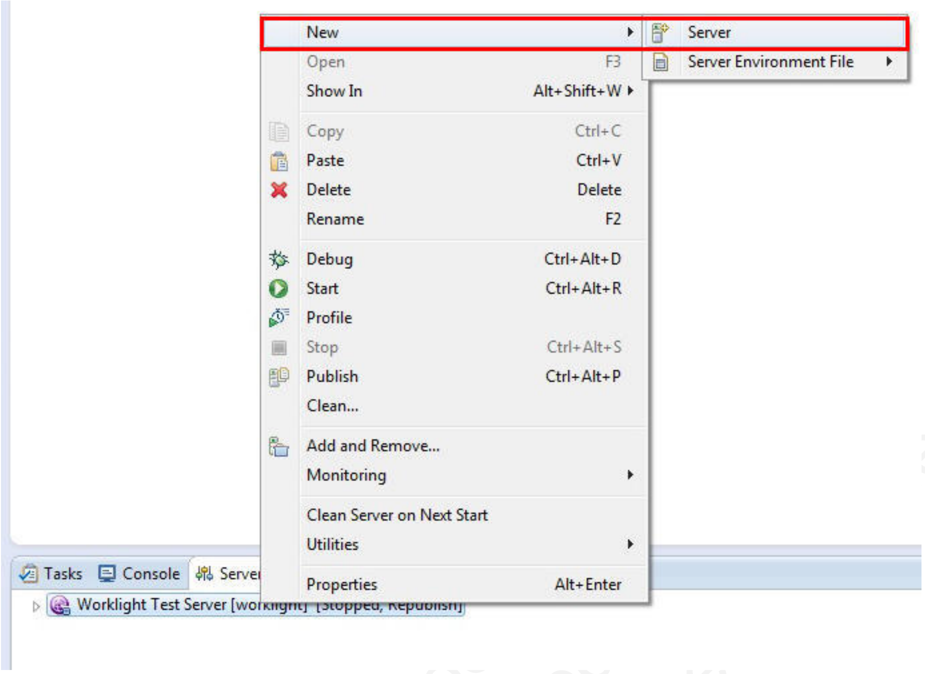
Server name: WebSphere Application Server V8.5 Liberty Profile at localhost

Server runtime environment: WebSphere Application Server V8.5 Liberty Profile

Download additional server adapters
type filter text
Apache
Basic
IBM
IBM WebSphere v6.0
WebSphere Application Server V8.5 Liberty Profile
JBoss
ObjectWeb
Lightweight profile for WebSphere Application Server.
Add...
Configure runtime environments...
< Back
Next >
Cancel
Finish

5. The new external server now is displayed in the **Server** field of the Configure Worklight Build and Deploy Target window. If you select the new server on the Configure Worklight Build and Deploy Target window, it becomes the default target test server, and all builds, deployments, and updates of the project WAR files made using the **Run As** > **Run on Worklight Development Server** command will go there.

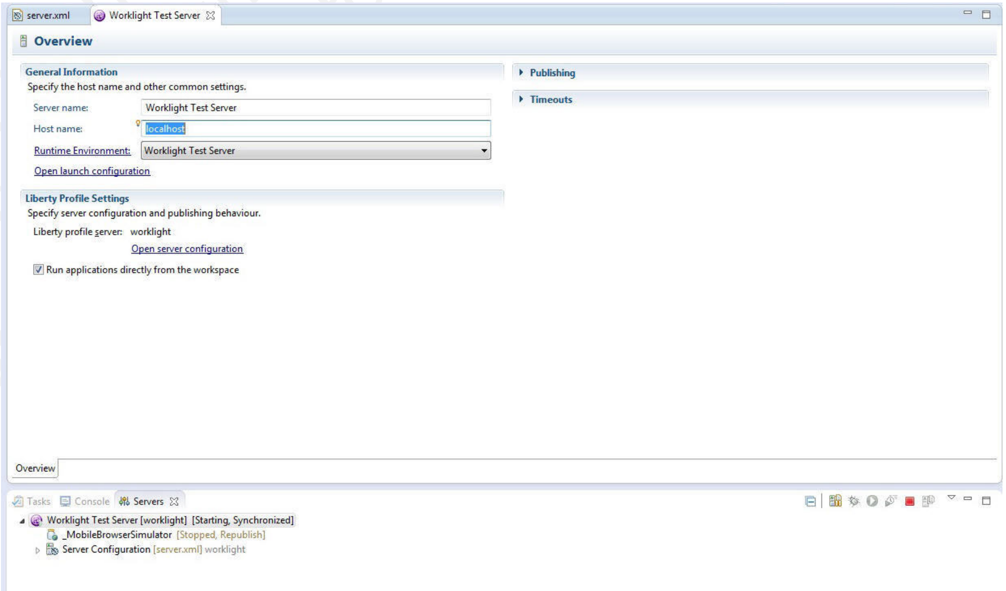
An alternate method of reaching this New Server window is to right-click the entry for an existing server in the Eclipse Servers view and select **New** > **Server** from the menu, as shown in the following screen capture:



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.



General Information

Specify the host name and other common settings.

Server name: Worklight Test Server

Host name: localhost

Runtime Environment: Worklight Test Server

Open launch configuration

Liberty Profile Settings

Specify server configuration and publishing behaviour.

Liberty profile server: worklight

Open server configuration

☑ Run applications directly from the workspace

Worklight Test Server [worklight] [Starting, Synchronized]

_MobileBrowserSimulator [Stopped, Republish]

Server Configuration [server.xml] worklight

For WebSphere Application Server Liberty Profile:

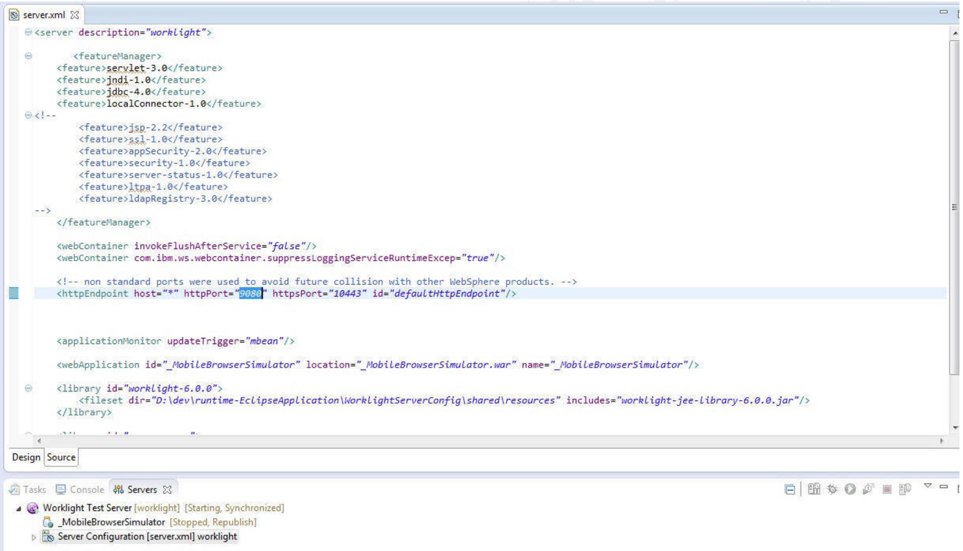
When you connect a project in Worklight Studio to an existing Liberty profile server (not the Worklight Development Server), you must check one thing before you attempt to build and deploy Worklight applications:

- 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 description="worklight">

        <featureManager>
    <feature>servlet-3.0</feature>
    <feature>jndi-1.0</feature>
    <feature>jdbc-4.0</feature>
    <feature>localConnector-1.0</feature>
<!--
        <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>ltpa-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-jee-library-6.0.0.jar"/>
    </library>
```

```
Worklight Test Server [worklight] [Starting, Synchronized]
    _MobileBrowserSimulator [Stopped, Republish]
  Server Configuration [server.xml] worklight
```

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. On 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.

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.

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 WL.OptionsMenu.

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.

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 “Fixing the Tab Bar on the screen – Android 2.2 and higher” and WL.TabBar.

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 27.

Fixing the Tab Bar on the screen – Android 2.2 and higher:

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 higher, 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, minim
```

2. Update the Android CSS BODY tag to also apply to the HTML tag, as follows:

```
html, body {
  height: auto;
  overflow: auto;
}
```

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 advances 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 356.
- 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 383.
- Use predefined application templates to speed up the development of your application. For more information, see “Mobile patterns” on page 394.
- Use all the power of Dojo Mobile through the Dojo library project. For more information, see “Working with the Dojo Library Project that serves Dojo resources” on page 358.
- 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 635.

- 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 371.

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.

Note: jQuery Core is provided in the Worklight Library.

You can download the required jQuery Mobile components, which are in the `.js` and `.css` files, at http://jquerymobile.com/download/. Download the zip file, which has a version number as part of the file name, for example `jquery.mobile-1.3.2.zip`. New versions of jQuery are released frequently.

Note: The tools require the non-minified version of the scripts (if necessary, replace anything with a “min” segment in the file name with the corresponding “full” file).

1. Create a Worklight project.
2. Right-click the project and select Hybrid Application.
3. Name the application and configure.
4. Browse for the folder where you downloaded the `jquery.mobile-Version.zip`.

From the populated selection, choose the required jQuery Mobile components, as follows:

- `jquery.mobile-Version.css`, contains all the styling for the mobile widgets and framework
- `jquery.mobile-Version.js`, the jQuery mobile framework
- `images`, which is the whole folder of images that are used by the style sheet for jQuery's built-in icons

If your project is already created, go ahead and create an application.

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 383.

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. Click **Configure javaScript Libraries**.
6. In the **Dojo installation** section, select **Add Dojo Toolkit** to add the Dojo facet and Dojo support to the application. When you build a mobile web application, Dojo is included to create the native application, such as an iPhone or Android application.
7. 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
Create a Dojo library project	1. Click **New Dojo Library** to open the Dojo Library Setup wizard. 2. In the **Name** field, enter a name for your new Dojo library project. 3. Specify the version of Dojo that you want to install. 4. Configure how your Dojo library project accesses the Dojo Toolkit and which version of the toolkit to use: • Click **Provided** to select a Dojo Toolkit that is provided with the product. • Click **On Disk** and then choose one of the following options: – Click **Archive File** to select an archive file of a compressed Dojo distribution. When you click **Finish**, the contents of the archive file are automatically extracted into your project. – Click **Folder** to browse to the root Dojo folder in another project in your workspace. 5. Expand the **Select the Dojo components to be included in the project** section and select the **Dojo components** that you want to include in your project. 6. Click **Finish**. The new project is now displayed as an option in the list of available Dojo library projects.

8. Click **Finish**. Both the Worklight project and the Dojo library project are created.

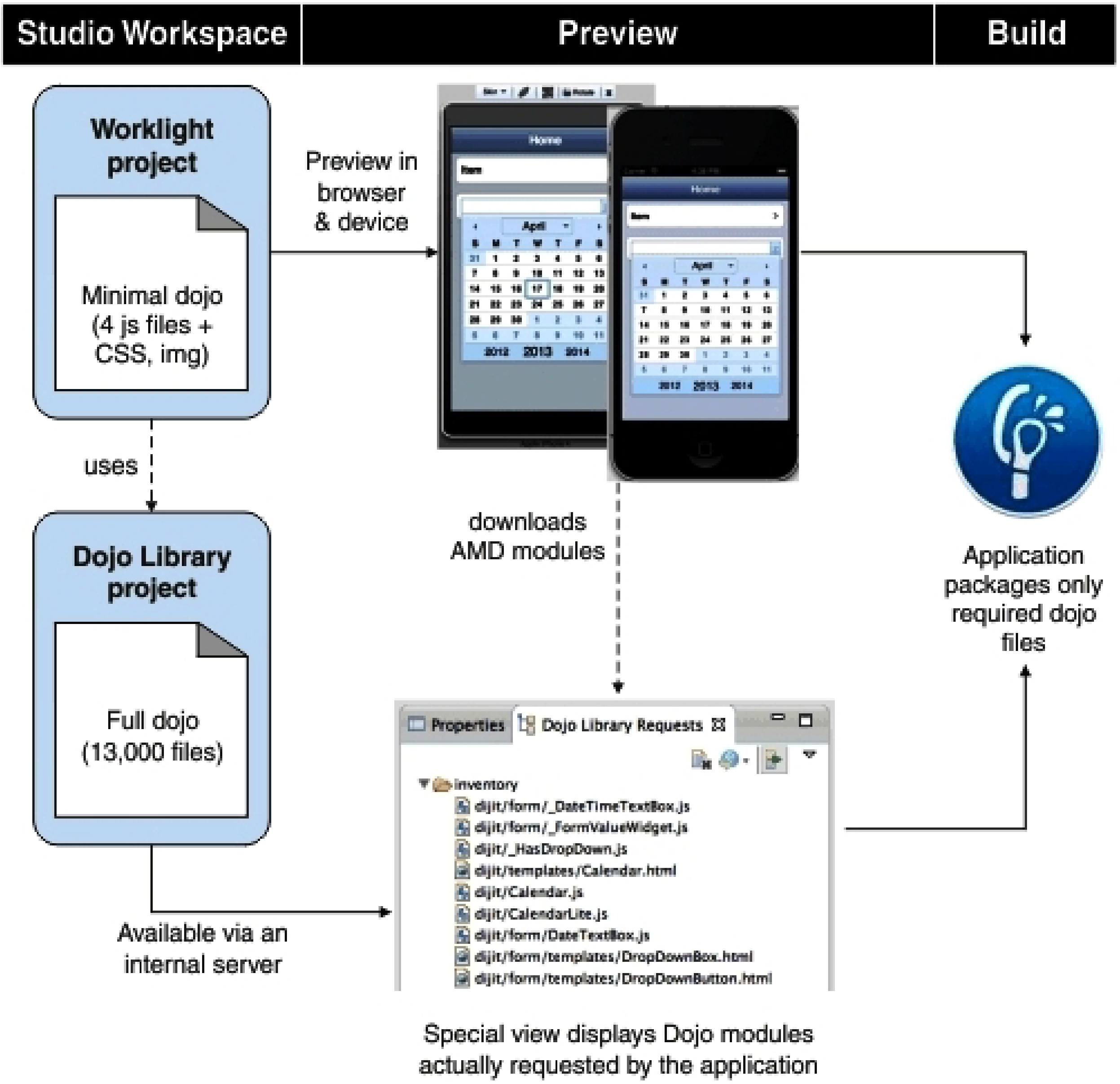
Working with the Dojo Library Project that serves Dojo resources:

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.

The Dojo library project contains a full distribution of Dojo. 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. The Dojo library project contains a full Dojo that you can use in a Dojo application. It is provided on an internal server, separate from your IBM Worklight application.

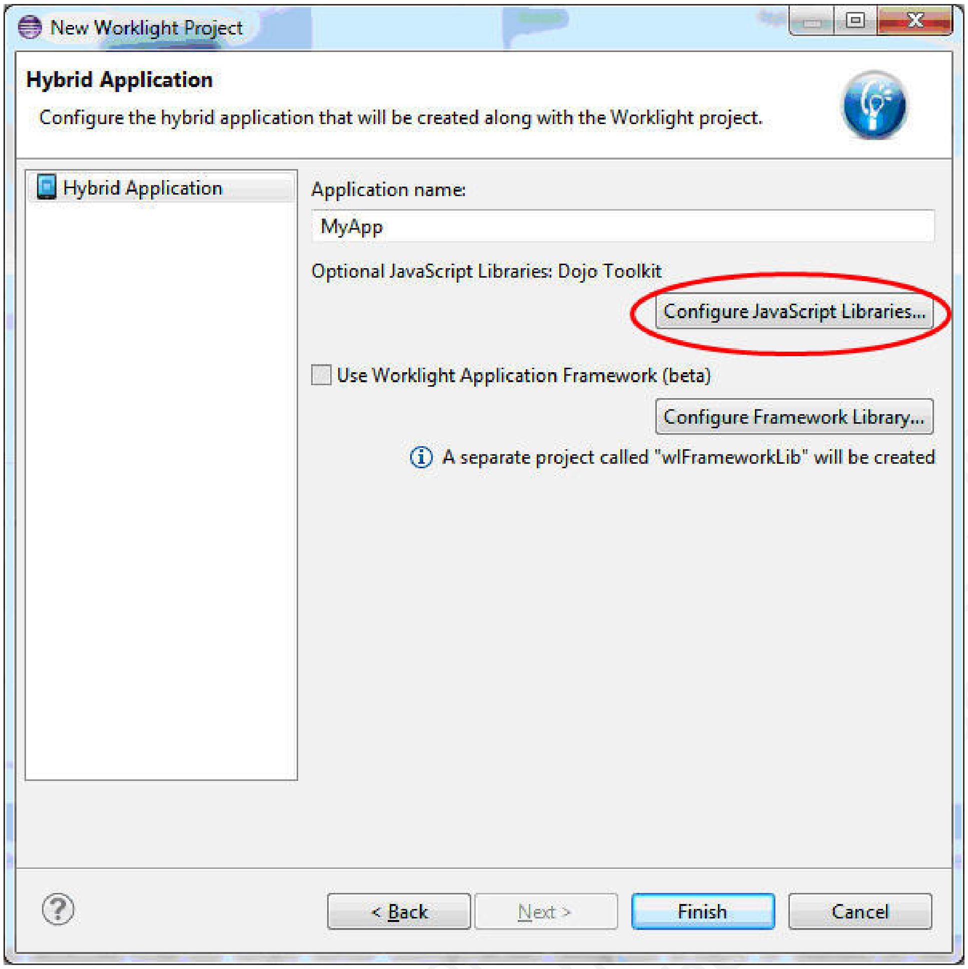
An IBM Worklight project is initialized with only a minimal set of mobile layers and themes. It contains the Dojo resources that are deployed as part of the Worklight application. The Dojo that is contained in the Worklight project is optimized for size and includes only the features that are required for a basic mobile application.

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. The Dojo library project, running separately on an internal server, provides faster builds, smaller projects, and accurate lists of the Dojo files that your application is requesting. Here is a view of the improved Dojo workflow for application size and development speed:


Studio Workspace
Preview
Build
Worklight project
Minimal dojo (4 js files + CSS, img)
Preview in browser & device
uses
Dojo Library project
Full dojo (13,000 files)
downloads AMD modules
Application packages only required dojo files
Available via an internal server
Properties
Dojo Library Requests
Special view displays Dojo modules actually requested by the application

```
inventory
    dijit/form/_DateTimeTextBox.js
    dijit/form/_FormValueWidget.js
    dijit/_HasDropDown.js
    dijit/templates/Calendar.html
    dijit/Calendar.js
    dijit/CalendarLite.js
    dijit/form/DateTextBox.js
    dijit/form/templates/DropDownBox.html
    dijit/form/templates/DropDownButton.html
```

To demonstrate, here are illustrations that show the start of a new Worklight project. After you click **Configure JavaScript Libraries**, a wizard opens where you must click **Add Dojo Toolkit**. An extra field in the template is displayed called **New Dojo Library**.



New Worklight Project

Hybrid Application

Configure the hybrid application that will be created along with the Worklight project.

Hybrid Application

Application name: MyApp

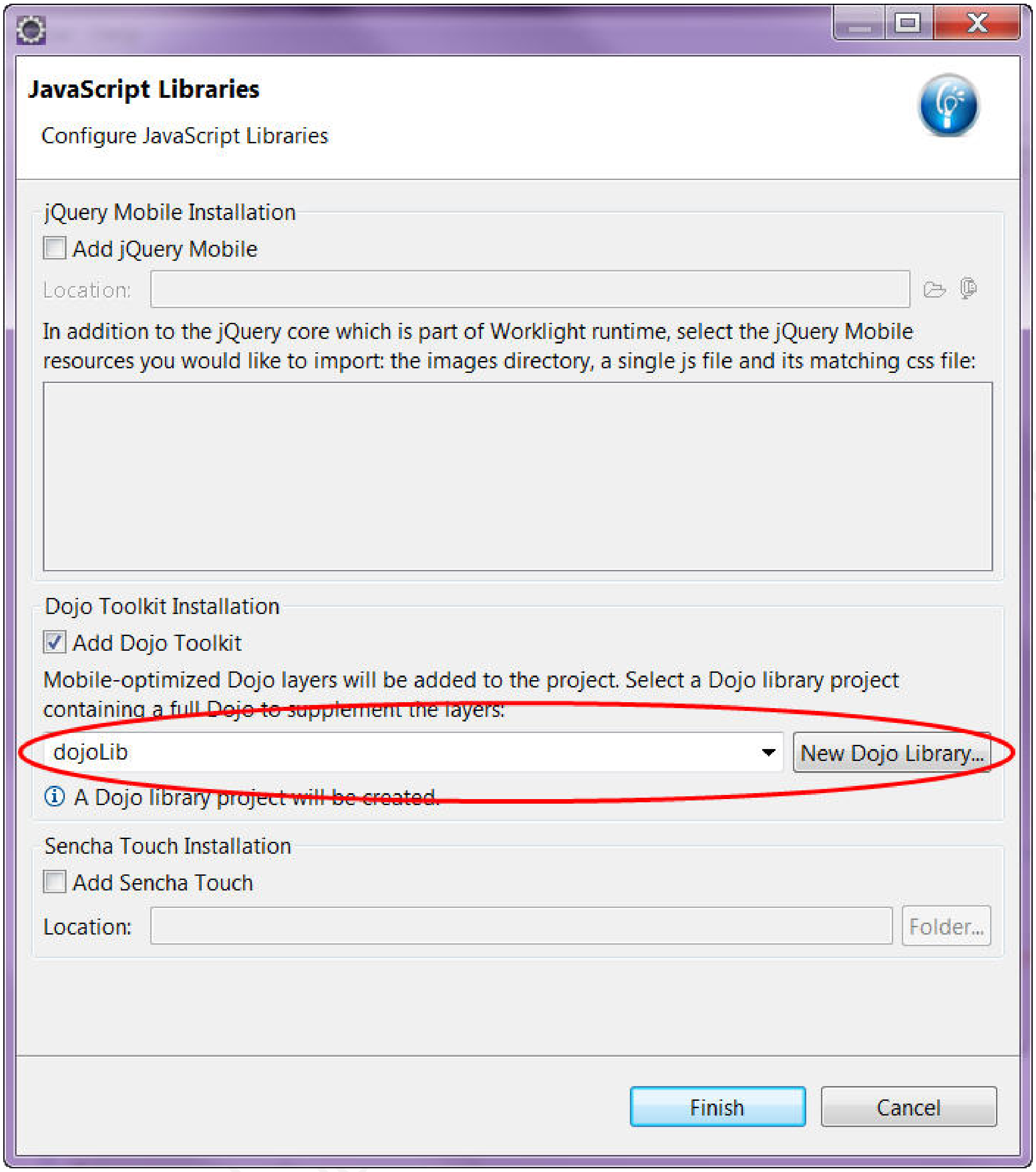
Optional JavaScript Libraries: Dojo Toolkit

Configure JavaScript Libraries...

□ Use Worklight Application Framework (beta)

Configure Framework Library...

A separate project called "wlFrameworkLib" will be created



JavaScript Libraries

Configure JavaScript Libraries

jQuery Mobile Installation

☐ Add jQuery Mobile

Location:

In addition to the jQuery core which is part of Worklight runtime, select the jQuery Mobile resources you would like to import: the images directory, a single js file and its matching css file:

Dojo Toolkit Installation

☑ Add Dojo Toolkit

Mobile-optimized Dojo layers will be added to the project. Select a Dojo library project containing a full Dojo to supplement the layers:

dojoLib ▾

ⓘ A Dojo library project will be created.

Sencha Touch Installation

☐ Add Sencha Touch

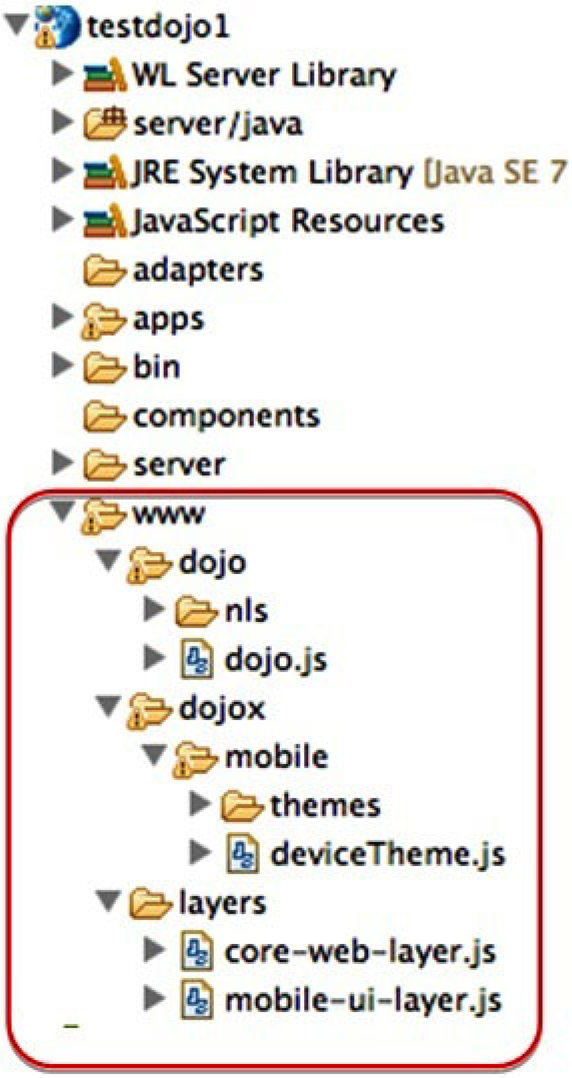
Location:

Note: You can create multiple libraries each for a different version of Dojo. A Worklight project is linked to one library.
The **New Dojo Library** feature is where access to the full distribution of Dojo is initiated by linking to the internal server. When you develop your application and test it, this feature supplies your **Dojo Library Requests** view with all the Dojo files requested during execution of your application. Select the files and move them to your project.

The minimum set of Dojo files that are provided in a Worklight project, are in a `www` folder in the navigation. It includes these files:

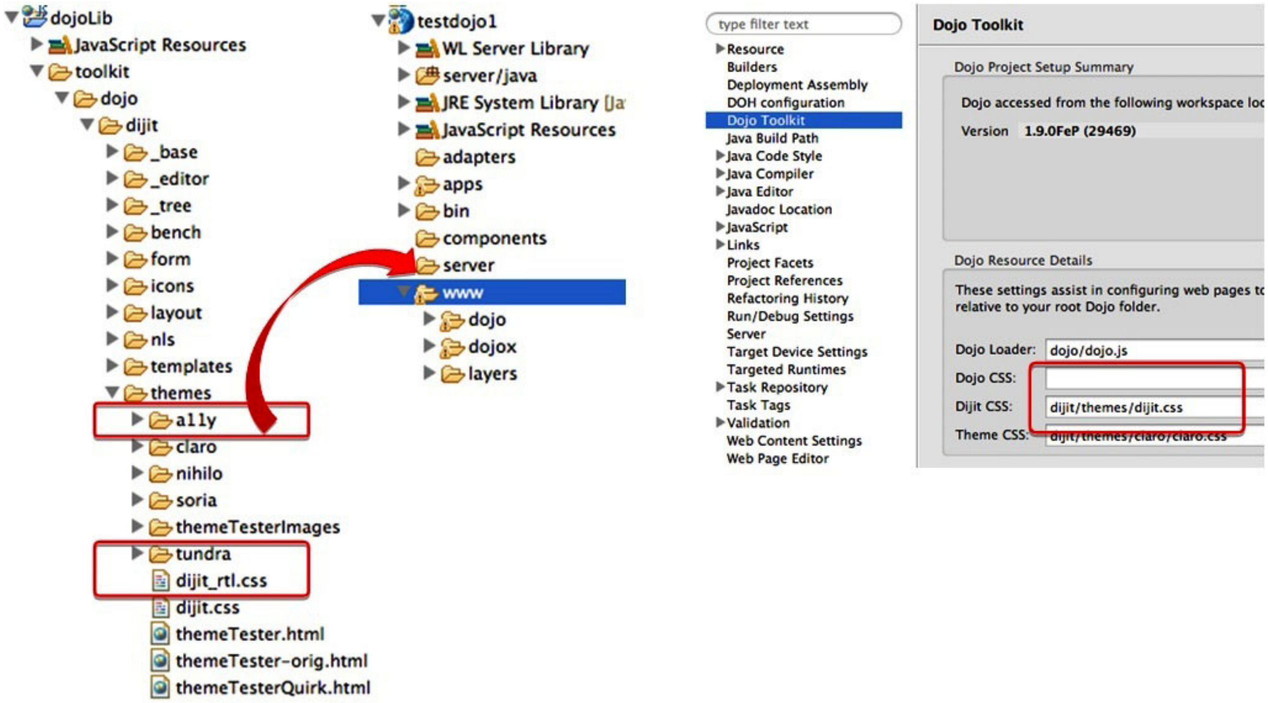
- Nano AMD loader (Dojo.js)
- Two layers for mobile widgets
- en-us NLS bundles for the two layers
- deviceTheme.js and mobile themes

With this formation, you can develop mobile pages by using Dojox mobile widgets.



```
▼ testdojo1
    ▶ WL Server Library
    ▶ server/java
    ▶ JRE System Library [Java SE 7
    ▶ JavaScript Resources
      adapters
    ▶ apps
    ▶ bin
      components
    ▶ server
    ▼ www
        ▼ dojo
            ▶ nls
            ▶ dojo.js
        ▼ dojox
            ▼ mobile
                ▶ themes
                ▶ deviceTheme.js
        ▼ layers
            ▶ core-web-layer.js
            ▶ mobile-ui-layer.js
```

You can manually set up more themes. First copy a theme into the www folder, and then set up the project css settings. Dijit widgets require a theme, Dojox widgets each bundle their own theme css.



Dojo Toolkit

Dojo Project Setup Summary

Dojo accessed from the following workspace loc

Version 1.9.0FeP (29469)

Dojo Resource Details

These settings assist in configuring web pages to relative to your root Dojo folder.

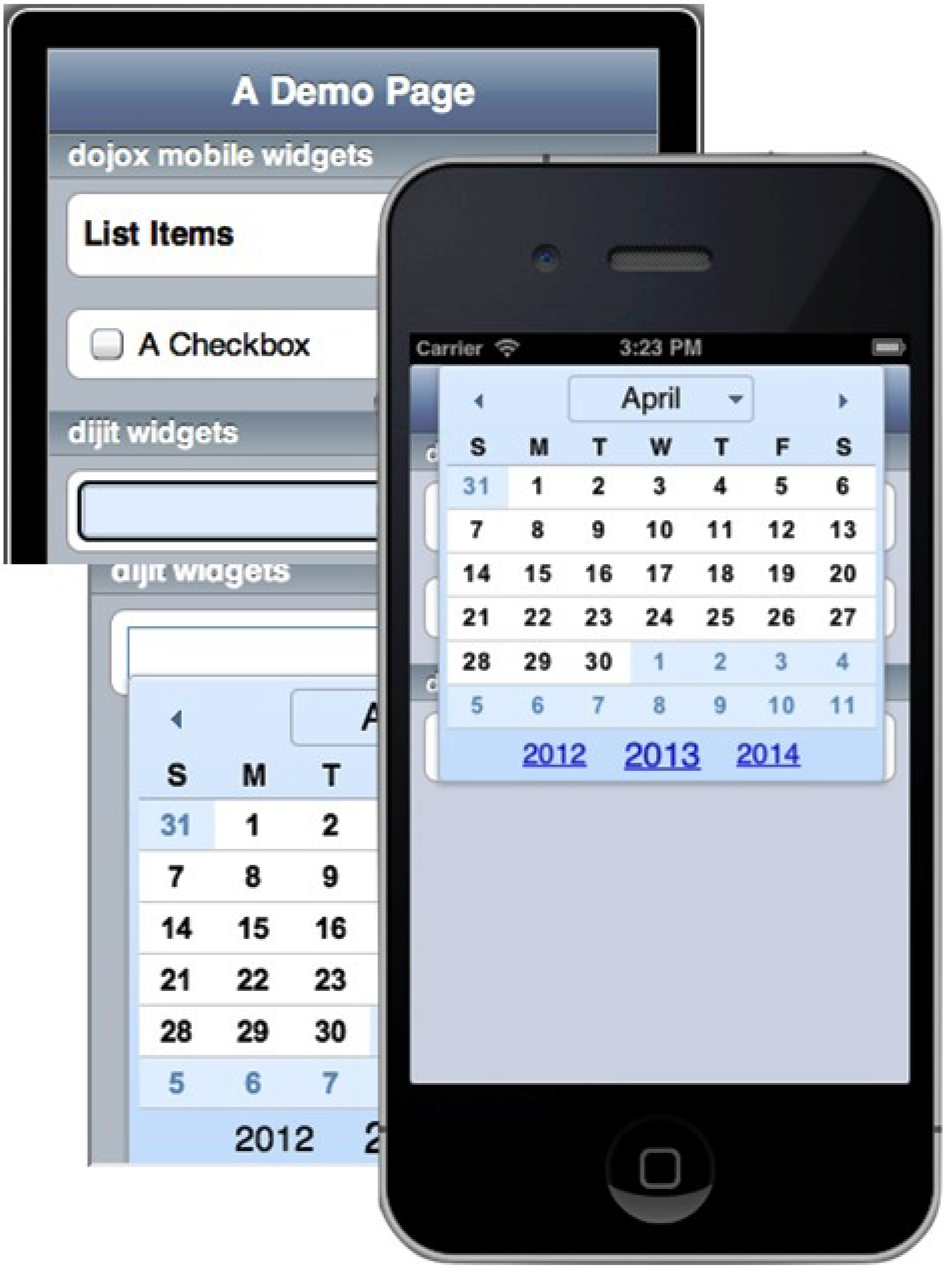
Dojo Loader: dojo/dojo.js

Dojo CSS:

Dijit CSS: dijit/themes/dijit.css

Theme CSS: dijit/themes/claro/claro.css

You can start coding your application. Use any Dojo modules (you no longer need to consider what files to import into the project). In this example, the “DateTextBox” comes from `dijit/form` but this module and its dependencies are not in the project yet.

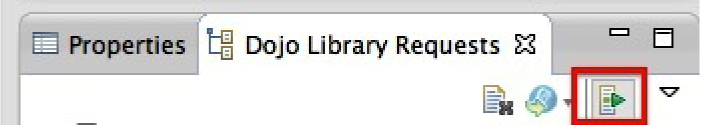


The Dojo loader is redirected to an internal server for modules in these packages: Dojo, dijit and Dojox.

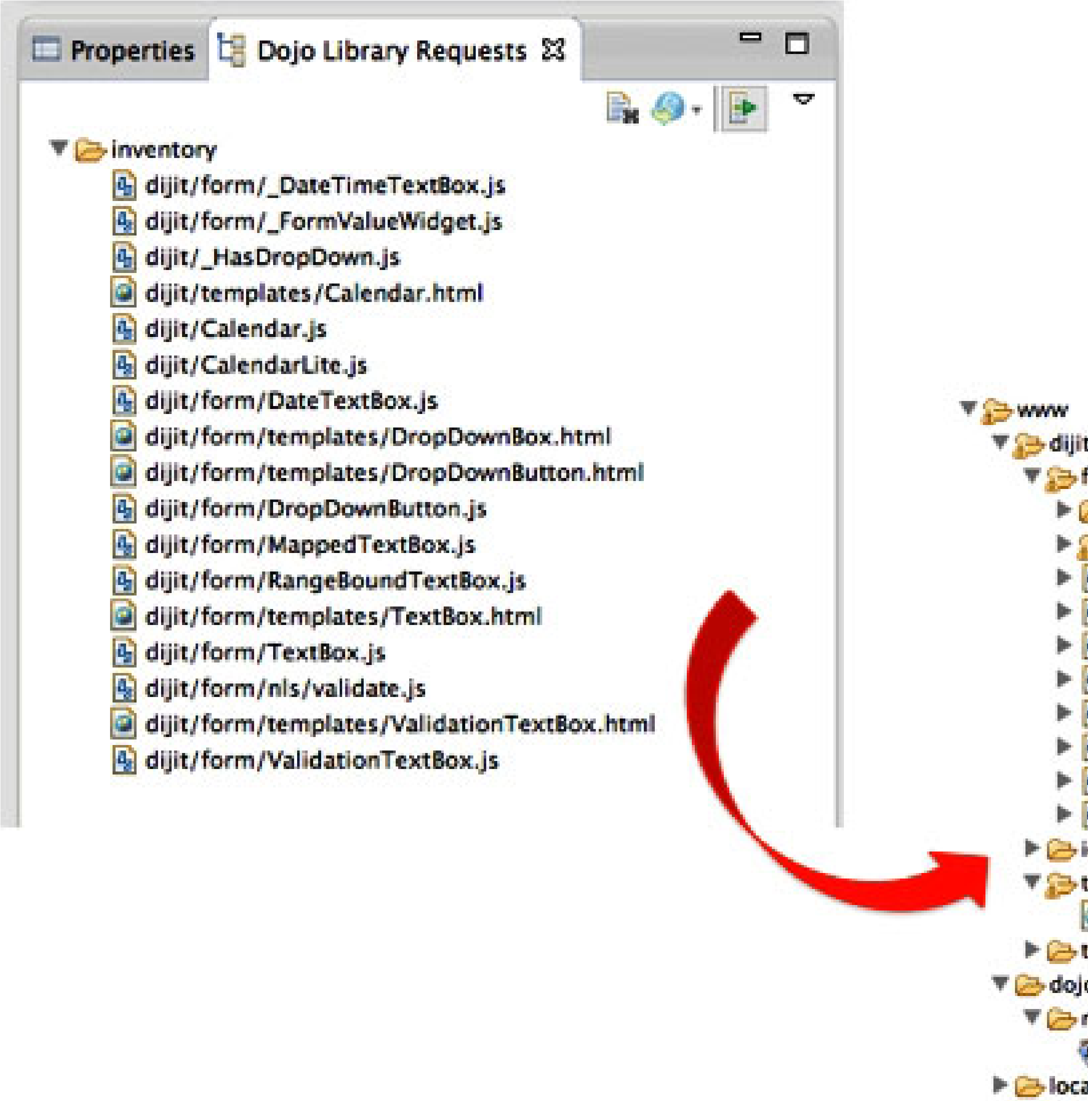
```
<script scr="worklight/cordova.js"></script>
<script scr="worklight/wljq.js"></script>
<script scr="worklight.js"></script>
<script scr="worklight/checksum.js"></script>
<script scr="worklight/plugins/org.apache.cordova.core.camera/www/ios/CameraPopoverHandle.js"></scri
<script scr="worklight/plugins/org.apache.cordova.core.contacts/www/ios/Contact.js"></script>
<script scr="worklight/plugins/org.apache.cordova.core.contacts/www/ios/contacts.js"></script>
<script scr="worklight/plugins/org.apache.cordova.core.file/www/ios/Entry.js"></script>
<script>windows.$ = window.jQuery = WLJQ;</script>
<script scr="http://192.168.0.100:9988/dojoLib/factory/inventory/iphone/dojo/dojo.js"> type="text/ja
```

Turn on the **Provide Missing Dojo Resources** function first, this action injects code to redirect the Dojo Loader to the server during the Worklight build.

To do this open **Dojo Library Requests** view and then click **Provide Missing Dojo Resources**, as illustrated here:

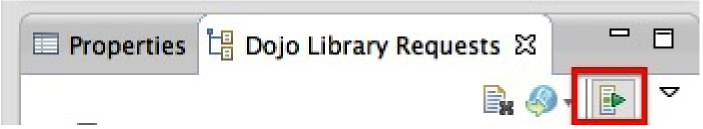


The special view output gives an accurate list of the Dojo files that are requested by the application. You can use this output as a guide to use the **Copy to Project** or **Copy to application** actions to copy files from the library into the project.



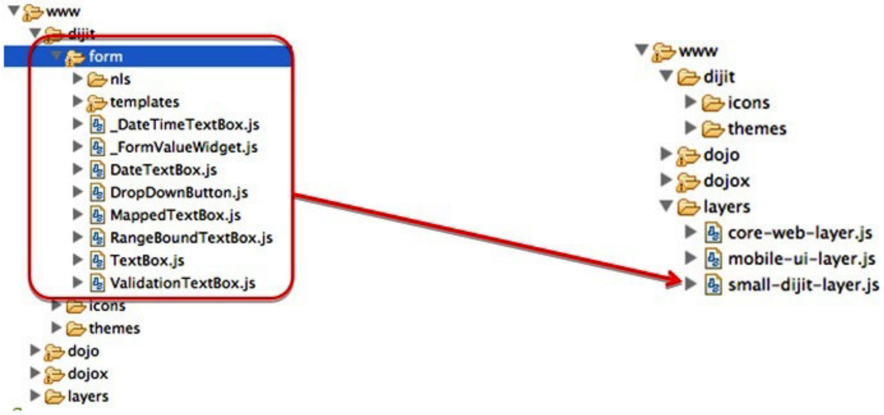
inventory
- dijit/form/_DateTimeTextBox.js
- dijit/form/_FormValueWidget.js
- dijit/_HasDropDown.js
- dijit/templates/Calendar.html
- dijit/Calendar.js
- dijit/CalendarLite.js
- dijit/form/DateTextBox.js
- dijit/form/templates/DropDownBox.html
- dijit/form/templates/DropDownButton.html
- dijit/form/DropDownButton.js
- dijit/form/MappedTextBox.js
- dijit/form/RangeBoundTextBox.js
- dijit/form/templates/TextBox.html
- dijit/form/TextBox.js
- dijit/form/nls/validate.js
- dijit/form/templates/ValidationTextBox.html
- dijit/form/ValidationTextBox.js

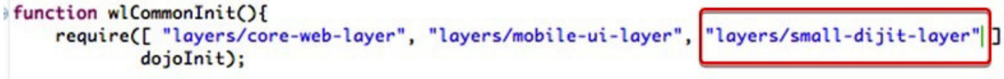
You must turn off the library and verify all the Dojo files are present in the application. Turn off the server



then, rebuild the environment and deploy (so that the injected Dojo loader config is removed) and then run the application again.

You can build the layers that are based on the code that is imported into the project (which is optional).

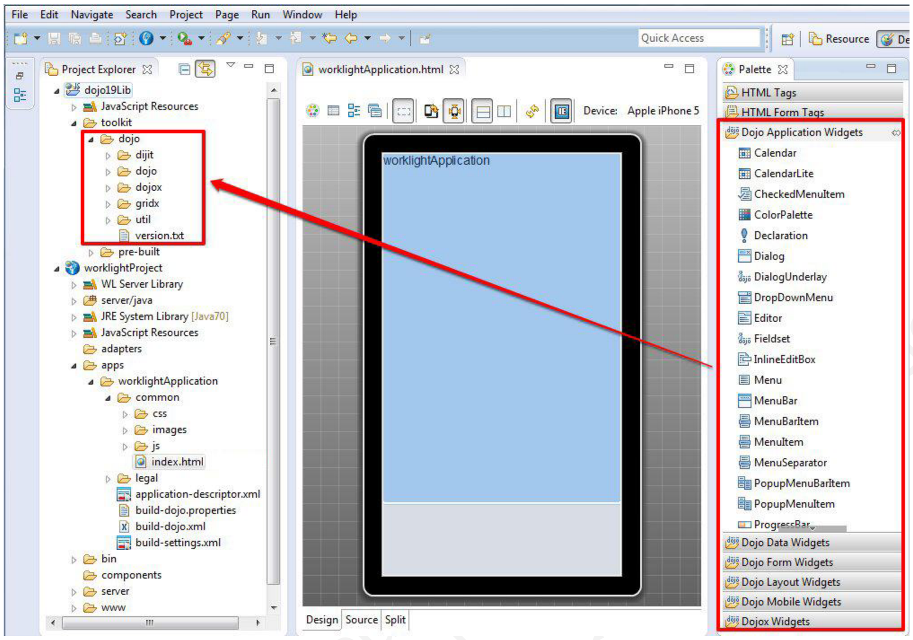




```
function wlCommonInit(){
    require([ "layers/core-web-layer", "layers/mobile-ui-layer", "layers/small-dijit-layer"]
            dojoInit);
```

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 content:

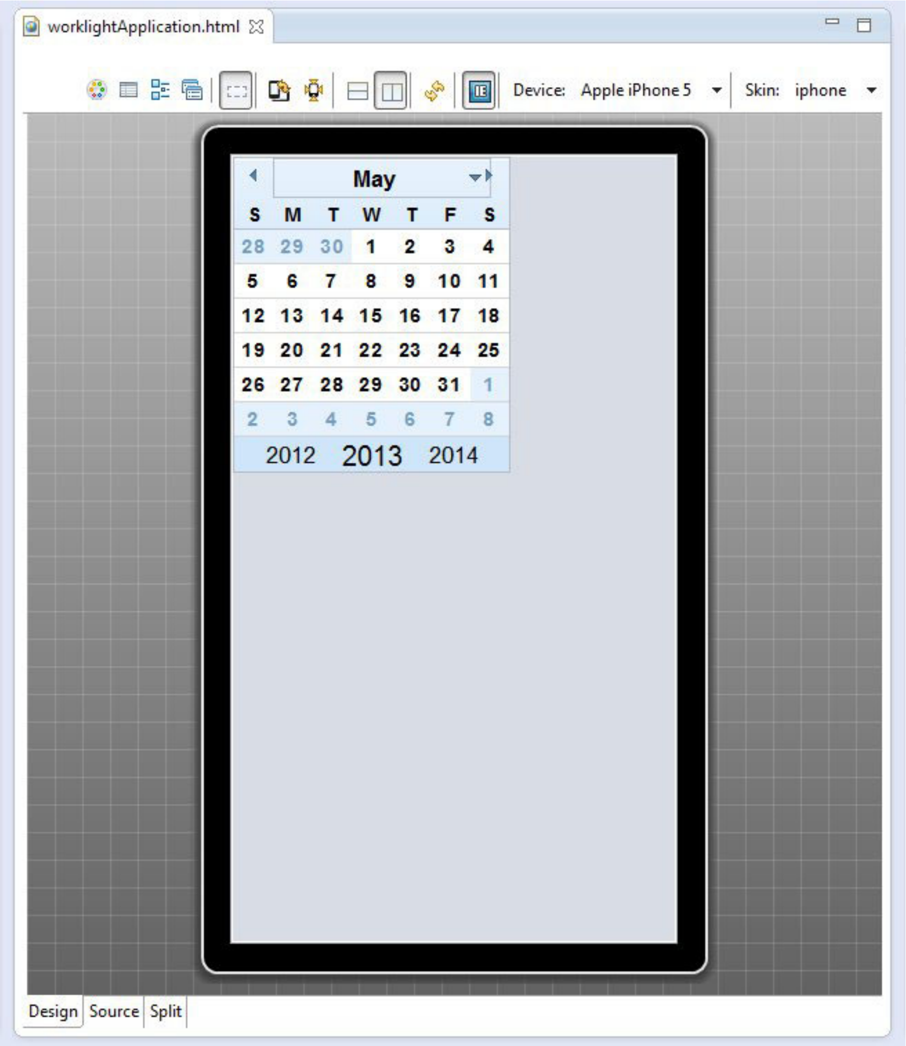
Rich Page Editor
The Rich Page Editor displays all of the widgets that are available in the Dojo library.



You can explore and test various Dojo artifacts. You could run and test your application outside of the Dojo library project. If you test on an external device or emulator, IDE must be running and must have Internet connectivity.

Worklight Studio provides the **Dojo Library Requests** view which shows what resources were requested from 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 to display this widget.

Note: If you run and test your application on a mobile device or use a device emulator, Eclipse must be running to provide Dojo Library resources. To shut down Eclipse and test your application in an environment that is similar to a production environment, you must remove Dojo Library instrumentation. See "Removing Dojo library instrumentation" on page 371.

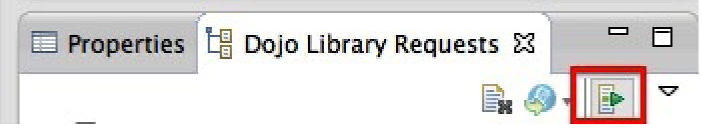


JavaScript source validation and content assist

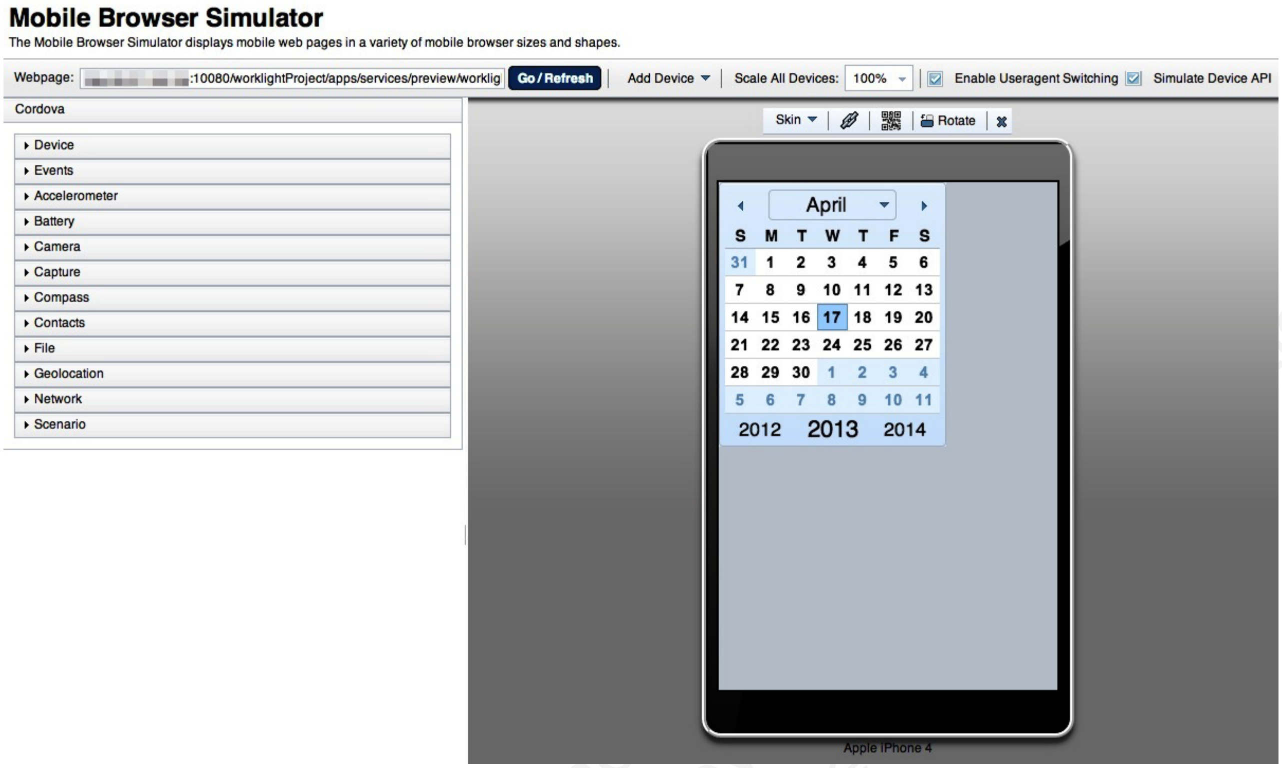
Content assist suggests all of the Dojo widgets that are contained in the Dojo library, or contained within the Worklight project, and new widgets that you have added to either of these. For example, if you have added your own Dojo widgets in either project, these new widgets will show up on the palette and in content assist.

Mobile Browser Simulator

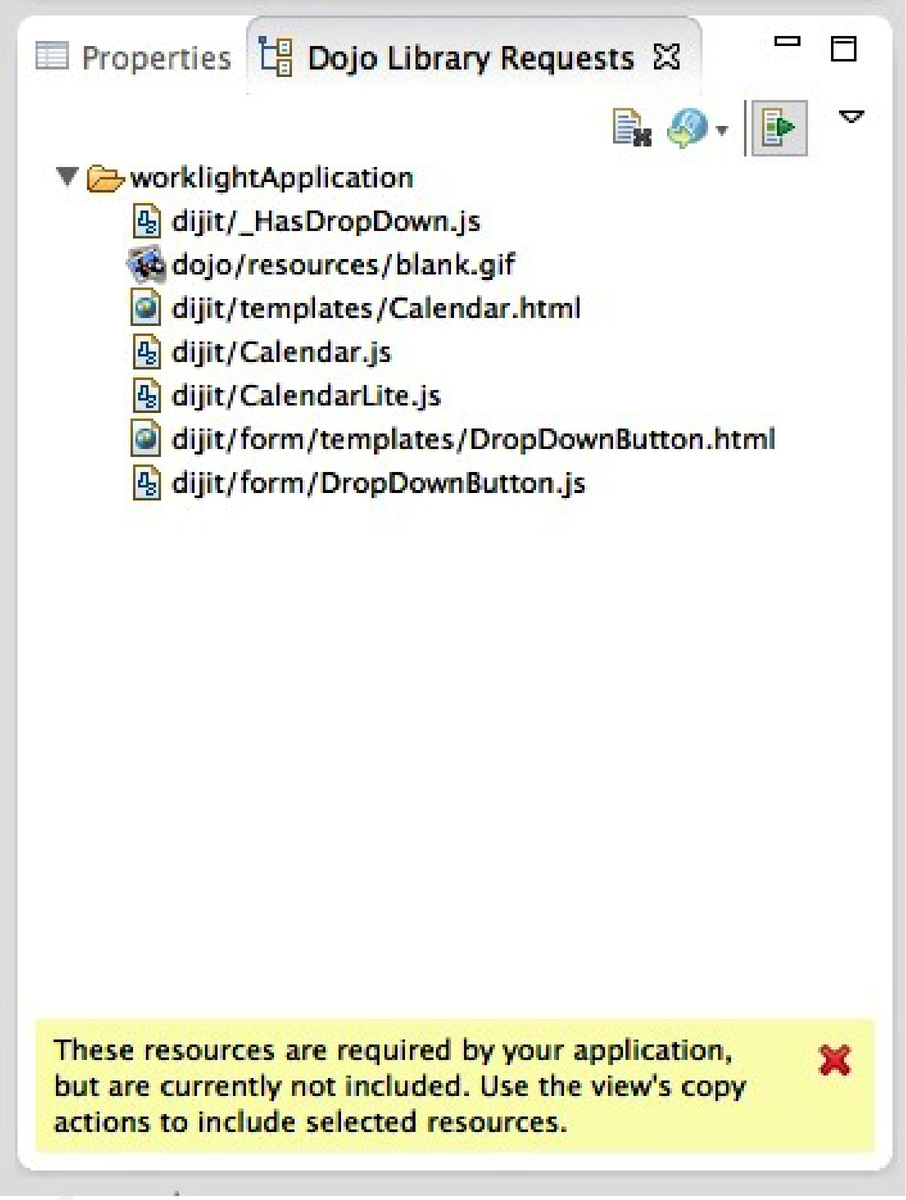
The Mobile Browser Simulator can run with or without the Dojo library resources. You can use the **Dojo Library Requests** view to turn on and off the Dojo library resources.



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** view shows which resources are served from the Dojo library project, which indicates the particular resources that are requested by the application but are not included as part of the application.



- worklightApplication
  - dijit/_HasDropDown.js
  - dojo/resources/blank.gif
  - dijit/templates/Calendar.html
  - dijit/Calendar.js
  - dijit/CalendarLite.js
  - dijit/form/templates/DropDownButton.html
  - dijit/form/DropDownButton.js

These resources are required by your application, but are currently not included. Use the view's copy actions to include selected resources.

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 Dojo Library Requests are not available outside of the Worklight Studio development environment.

To add the missing resources to your application, the view provides two copy actions.

The **Copy to Project** action copies selected resources into the project's `www` folder. Resources here are built into all Dojo-enabled applications in the project, which is useful when your applications use a common module or resource. The Copy to application action copies selected resources into the requesting application's common folder, which is useful when an application uses resources that are unique to that application.

If you disable 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 shown in the **Dojo Library Requests** view.

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:

"Working with the Dojo Library Project that serves Dojo resources" on page 358
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.

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, by using Dojo 1.8 layer files with a Dojo 1.9 library, may 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.

Implementing a different version of the Dojo Toolkit:

If you need to use a different Dojo Toolkit version, a special procedure is required.

Worklight Studio facilitates the integration of Dojo Toolkit into hybrid mobile applications. However, this Dojo Toolkit and its corresponding optimized resources (called Dojo layers) are tied to a fixed version per release, which is bundled within Worklight Studio.

Worklight Studio has a set of tools to facilitate the integration of latest available Dojo toolkit into a hybrid application. It also supplies a pre-built set of Dojo files (called Dojo layers) that bundle the Dojo Mobile modules in a few optimized resources. These files are copied by default into your Hybrid Dojo project under the `www` folder.

Note: Having these custom built layers is required for production deployments, not just for performance improvements but for a known limitation in Android environments. For more information, see the Dojo Toolkit website..

Even though Worklight Studio provides all the necessary resources to work with the most updated Dojo Toolkit, there is a chance that you might need to move to a different Dojo version or even modify the contents of the pre-defined layers. You also need to optimize your resources for production deployment.

Here is described how those layers are built and the necessary changes that need to be done in order to have a fully working application with a modified or updated Dojo toolkit.

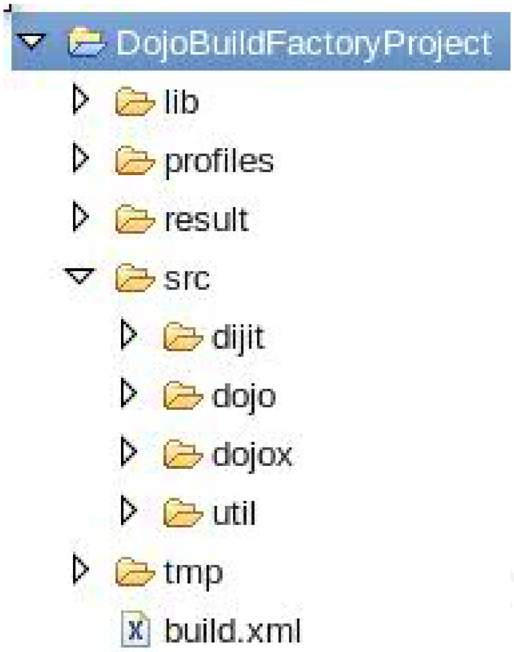
Building standard Dojo layers:

Here are the steps for how to build the standard Dojo layers.

Procedure

1. Download the latest version of the Dojo-Build-Factory build tool.
   a. Go to https://github.com/pruzand/Dojo-Build-Factory
   b. Select the branch for the Dojo version you are using in your project.
   c. Download the compresed repository from the website, which is usually a file with a name similar to `Dojo-Build-Factory-Dojo-Version.zip`
   d. Extract the file to a known location on your system.

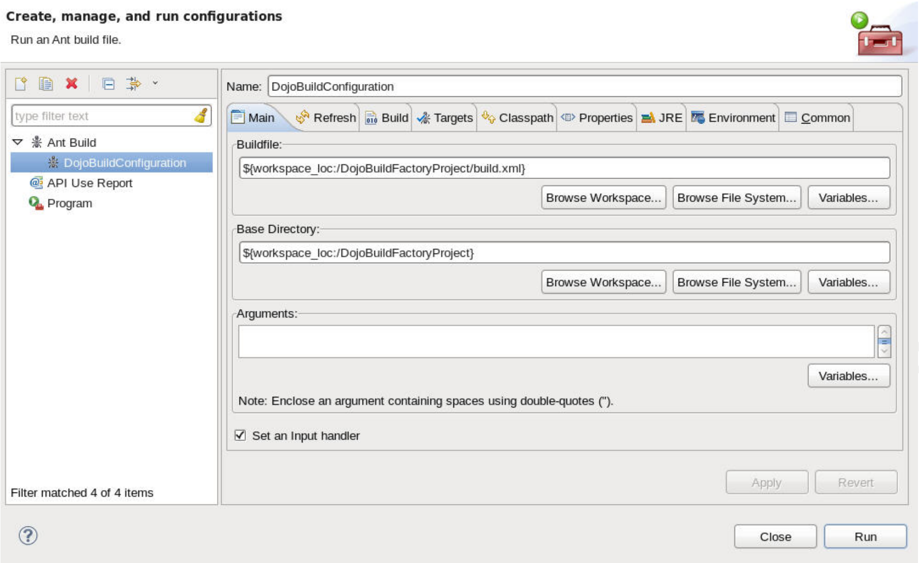
   Alternatively, if you Git is installed on your system, you can clone the Dojo-Build-Factory branch for the Dojo version you need, entering the following command from your system's command line: **`git clone -b <Dojo-Version-Branch> https://github.com/pruzand/Dojo-Build-Factory.git`**
   After you have the Dojo-Build-Factory build tool, a directory structure is created that contains the following files and folders:
   - build
   - releases
   - LICENSE
   - README
2. Open Worklight Studio and create a simple project:
   a. **File** → **New** → **Other** → **General** → **Project**
   b. Give project a valid name, such as `DojoBuildFactoryProject`.
   c. Click finish.
3. Copy and paste the contents of the Dojo-Build-Factory/build folder to your new project root.
4. Add the Dojo source from which you want to generate your optimized version to the `src` folder of the project. The Dojo source must be a full, decompressed source release of the Dojo Toolkit. Therefore, it must contain the `util` folder, since it is used for the layers generation. The project now has the following structure and files:



5. Open `profiles/env-config.js` and change the **`localeList`** to specify the relevant locales you want to include. By default, Dojo Build Factory includes only US English as the default language, so you see the following entry: `"localeList": "en-us"`. If you want to specify additional locales, you can set this variable to something like: `"localeList" : "en-us,ar,az,ca,cs,da,de,el,es,fi,fr,he,hr,hu,it,ja,kk,ko,nb,nl,pl,pt,pt-pt,ro,ru,sk,sl,sv,th,tr,zh,zh-tw"`

   Note: If the language setting for the target mobile phone is expected to not be US English, then you must specify that language in this **`localeList`**. Otherwise, it is highly possible that your application will not work.

6. Run the Dojo Mobile build.
   a. From within Worklight Studio, navigate to **Run** → **External Tools** → **External Tools Configuration**.
   b. Select **Ant Build** and click **New launch configuration**.
   c. Set the **Buildfile** field to point to the `build.xml` in the root of your Dojo build factory project.
   d. Set the **Base Directory** to your Dojo build factory project root. Here is an example of what your configuration could look like.:



Create, manage, and run configurations

Run an Ant build file.

Name: DojoBuildConfiguration

Buildfile: ${workspace_loc:/DojoBuildFactoryProject/build.xml}

Base Directory: ${workspace_loc:/DojoBuildFactoryProject}

Arguments:

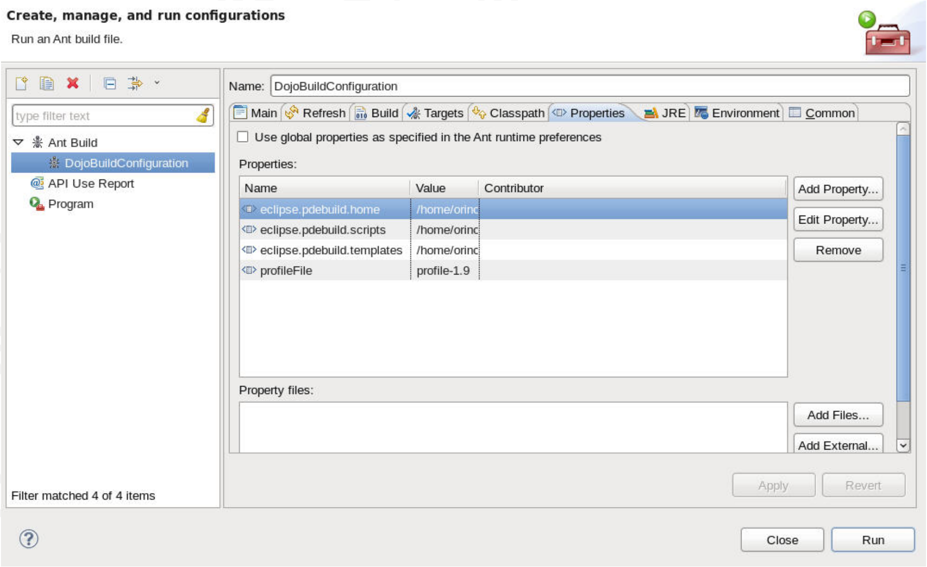
Note: Enclose an argument containing spaces using double-quotes (").

☑ Set an Input handler

Filter matched 4 of 4 items

e. Go to the Properties tab and confirm that **Use global properties as specified in the Ant runtime preferences** is cleared.

f. Add a new property with **Name** set to `profileFile` and value set to the name of the profile file that is located in your `project/profiles folder` without the `.js` extension. For example, `profile-1.9`



Create, manage, and run configurations

Run an Ant build file.

Name: DojoBuildConfiguration

□ Use global properties as specified in the Ant runtime preferences

Properties:

Name	Value	Contributor
eclipse.pdebuild.home	/home/orinc	
eclipse.pdebuild.scripts	/home/orinc	
eclipse.pdebuild.templates	/home/orinc	
profileFile	profile-1.9	

Property files:

Filter matched 4 of 4 items

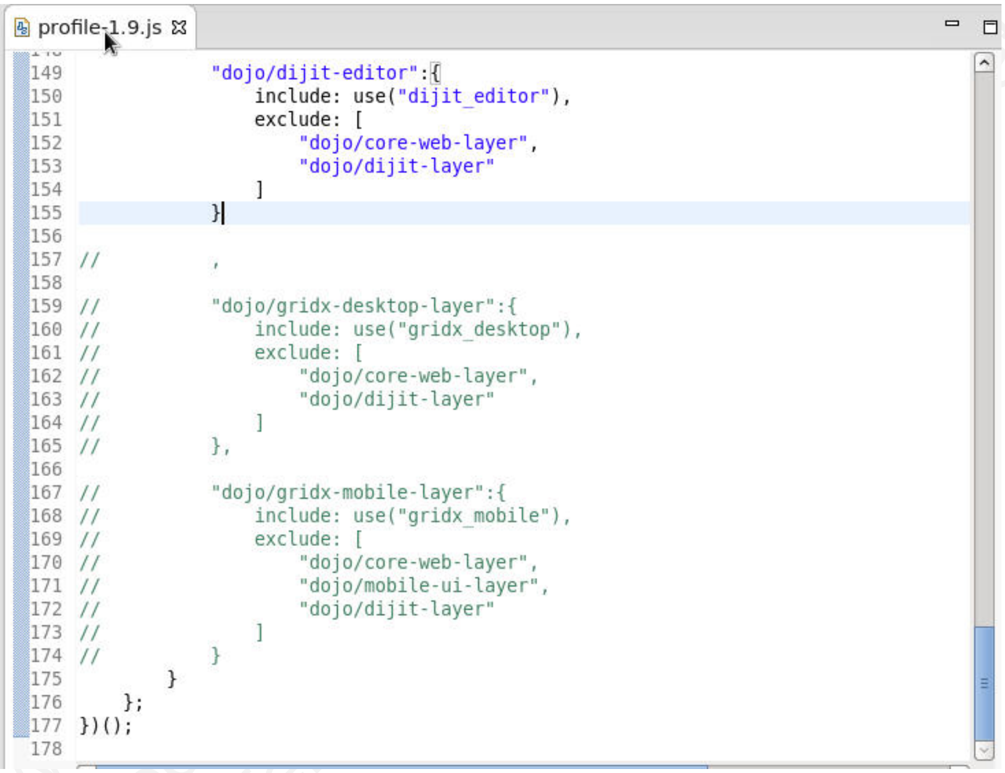
g. After you set up the configuration, click **Run**. The build takes about 6-15 minutes and can be monitored in the console view.

Results

After completion, the generated Dojo layer files are located inside your project in the `result/compressed/dojo` directory, following the `*-layer.js` naming convention. If you cannot see those files, do a refresh (F5) on your project root. There are several different layer files available there, which is much more than just the `mobile-ui-layer.js`. You can find the details on what every layer contains on the Github website.. These instructions can only guide you through the process of replicating the structure that Worklight Studio provides. Depending on your

Worklight Hybrid application requirements, you might want to append extra layers to your project to make additional optimized Dojo resources available.

Note: There can be problems when you produce the Dojo 1.9 custom build. You might see errors such as: `error(303) Missing include module for layer. missing: gridx/allModules; layer: dojo/gridx-desktop-layer` To resolve these errors, navigate to `project/profiles`, open the`profile-version.js` file, then look for the failing layer(s) (in this case gridx-desktop-layer, but also gridx-mobile-layer fails) and comment out the corresponding layer(s) definition to pull it out from the custom build:



```
                "dojo/dijit-editor":{
                    include: use("dijit_editor"),
                    exclude: [
                        "dojo/core-web-layer",
                        "dojo/dijit-layer"
                    ]
                }

//              ,

//              "dojo/gridx-desktop-layer":{
//                  include: use("gridx_desktop"),
//                  exclude: [
//                      "dojo/core-web-layer",
//                      "dojo/dijit-layer"
//                  ]
//              },

//              "dojo/gridx-mobile-layer":{
//                  include: use("gridx_mobile"),
//                  exclude: [
//                      "dojo/core-web-layer",
//                      "dojo/mobile-ui-layer",
//                      "dojo/dijit-layer"
//                  ]
//              }
            }
    };
})();
```

What to do next

Repeat build execution until it finishes successfully.

Switching an existing project to a new Dojo library:

Here are the steps for moving to a different Dojo Library with the resources you created previously.

Before you begin

If you already have a Hybrid Dojo Project with a Hybrid app that points to an existing dojoLib project, which is the default, here is the process to move to a different Dojo Library.

Procedure

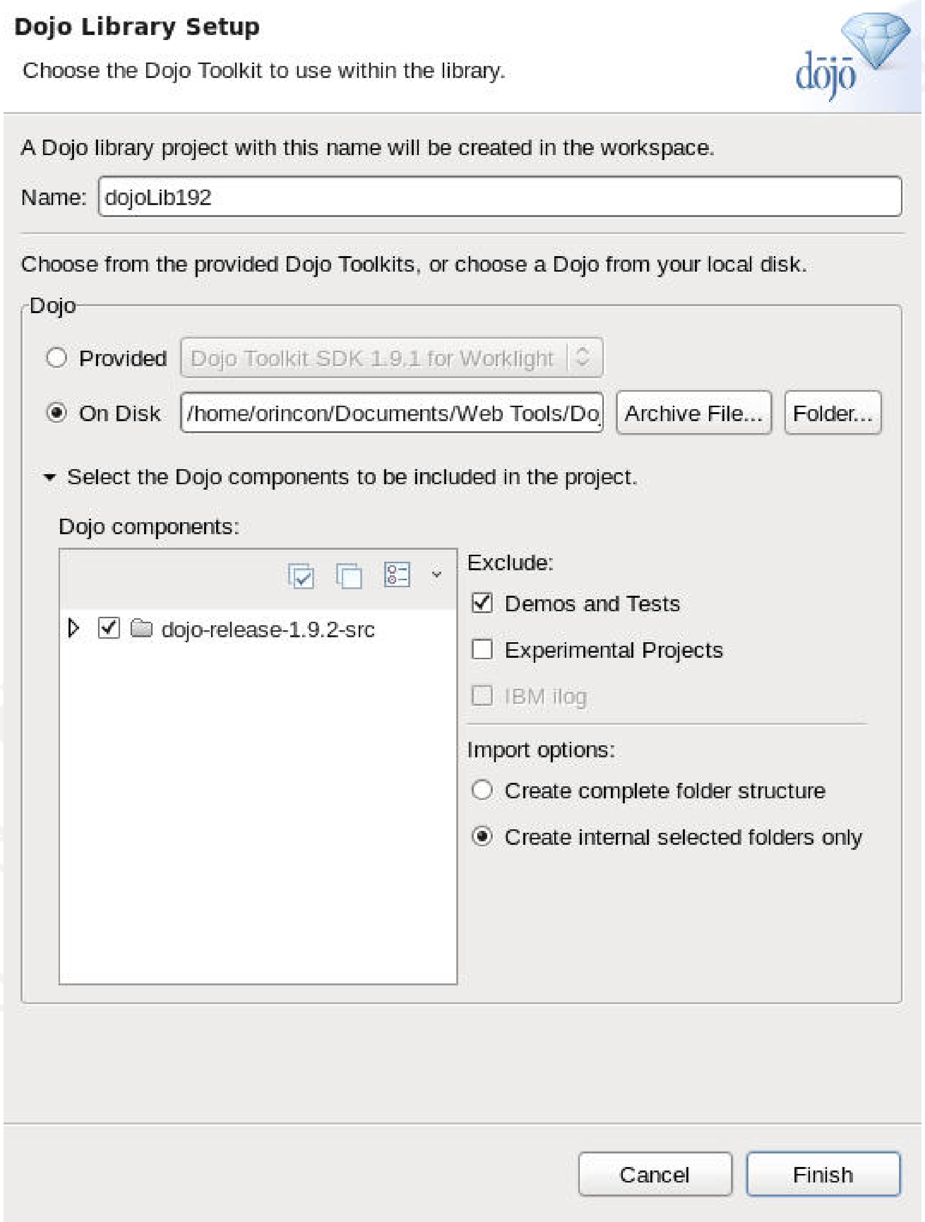
1. Switch the current Dojo project to a different Dojo Library.
   a. In Worklight Studio, right click **DojoProject**, then select **Properties** menu item.

b. Go to Dojo Toolkit properties page

c. Click **New Dojo Library**

d. Give the library a valid name. For example, `NewDojoLib`

e. Select **On Disk** and then fill it with the archive file or folder that contains the Dojo source. For example, `dojo-release-1.9.2-src.tar.gz` Use the same you previously used to create the Dojo Custom build. For the archive file, specify the **Create internal selected folder only** under the **Import options** section if available.



Dojo Library Setup

Choose the Dojo Toolkit to use within the library.

A Dojo library project with this name will be created in the workspace.

Name: dojoLib192

Choose from the provided Dojo Toolkits, or choose a Dojo from your local disk.

Dojo

○ Provided Dojo Toolkit SDK 1.9.1 for Worklight

● On Disk /home/orincon/Documents/Web Tools/Do

Select the Dojo components to be included in the project.

Dojo components:

☑ dojo-release-1.9.2-src

Exclude:

☑ Demos and Tests

□ Experimental Projects

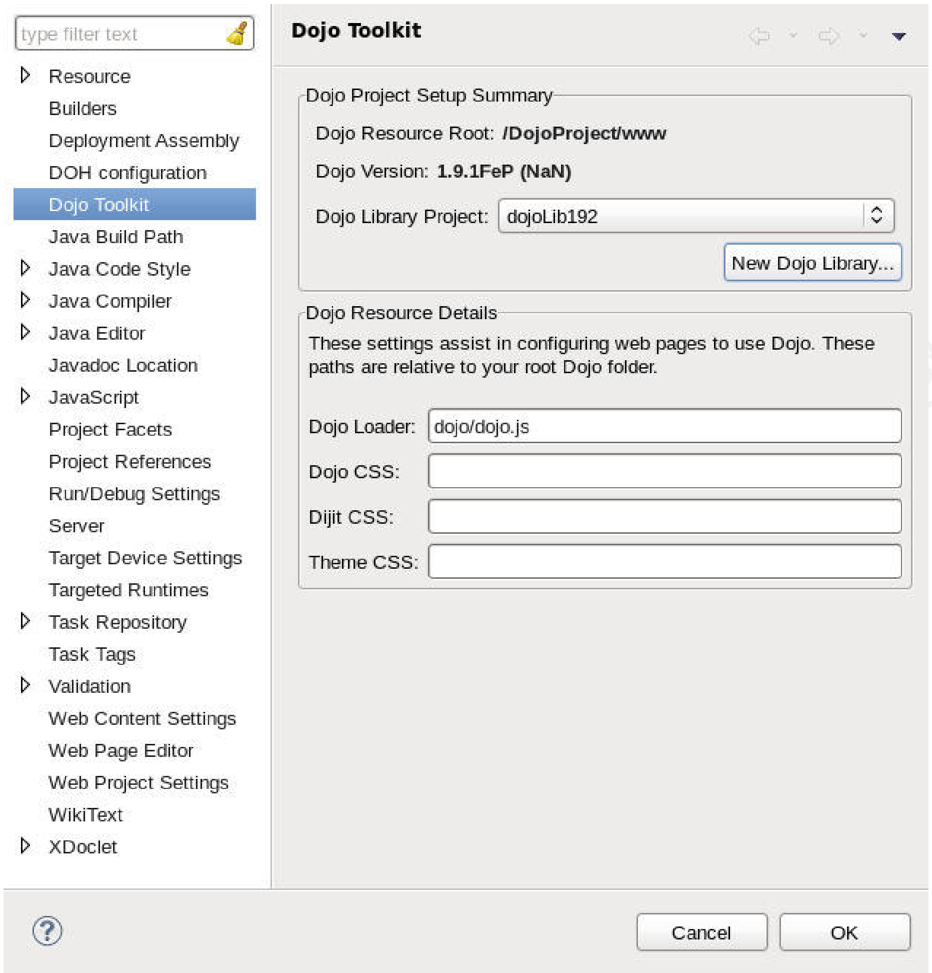
□ IBM ilog

Import options:

○ Create complete folder structure

● Create internal selected folders only

f. Click **Finish** to close this dialog. Confirm that the properties page indicates it uses the new Dojo Library, then click **OK**.



Dojo Toolkit

Dojo Project Setup Summary

Dojo Resource Root: **/DojoProject/www**

Dojo Version: **1.9.1FeP (NaN)**

Dojo Library Project: dojoLib192

Dojo Resource Details

These settings assist in configuring web pages to use Dojo. These paths are relative to your root Dojo folder.

Dojo Loader: dojo/dojo.js

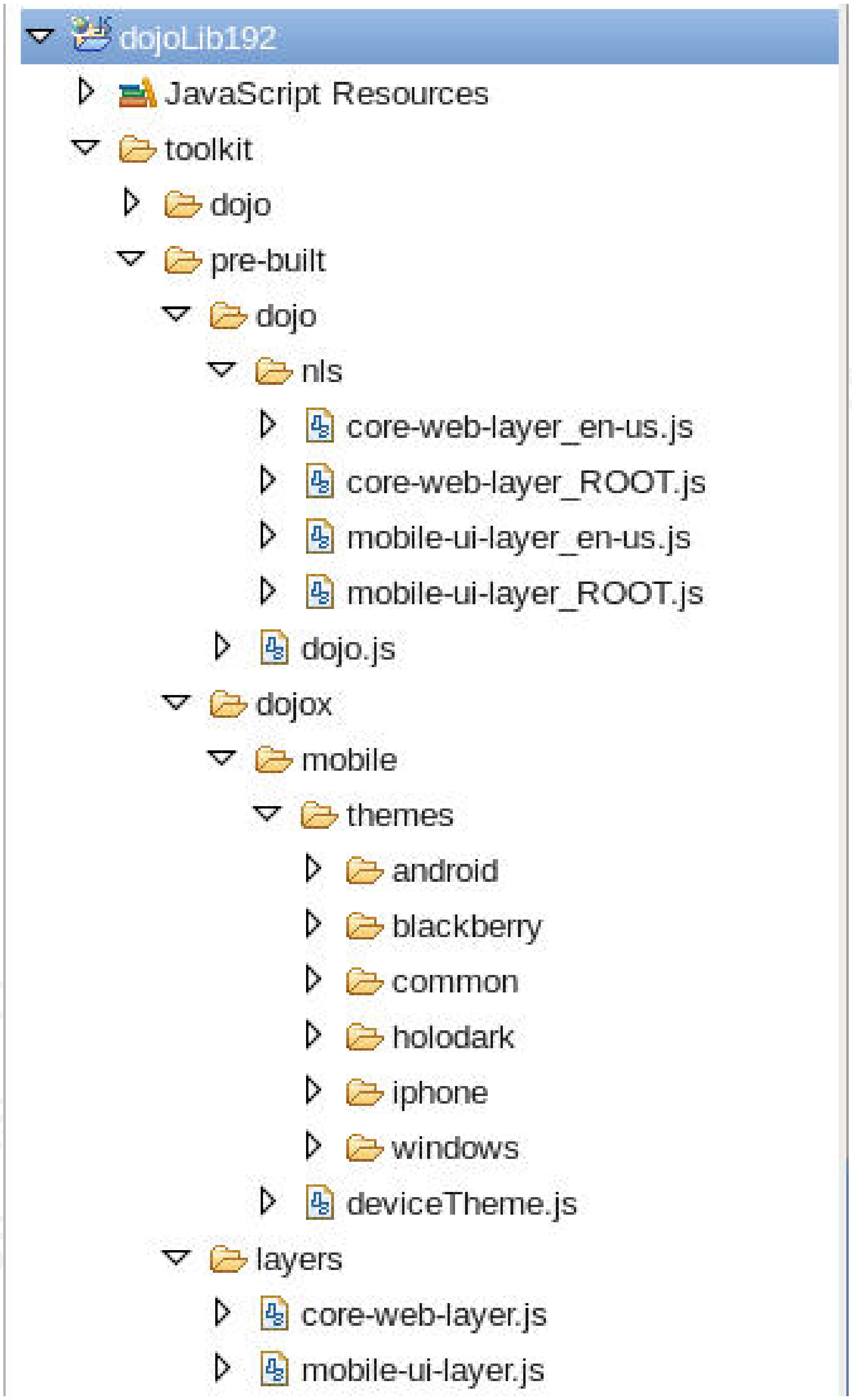
Dojo CSS:

Dijit CSS:

Theme CSS:

A new Dojo Library project must be created, but it is still incomplete because it lacks the optimized layers you created previously, so there are still few steps to complete.

2. Go to the newly created Dojo Library project and create a new folder that is called `pre-built` under the existing `toolkit` one.
3. Populate the pre-built folder by creating the following directory structure.



- dojoLib192
  - JavaScript Resources
  - toolkit
    - dojo
    - pre-built
      - dojo
        - nls
          - core-web-layer_en-us.js
          - core-web-layer_ROOT.js
          - mobile-ui-layer_en-us.js
          - mobile-ui-layer_ROOT.js
        - dojo.js
      - dojox
        - mobile
          - themes
            - android
            - blackberry
            - common
            - holodark
            - iphone
            - windows
          - deviceTheme.js
      - layers
        - core-web-layer.js
        - mobile-ui-layer.js

Copy the files from the `result/compressed` folder in the Dojo Build Factory project.

Updating the optimized resources in the Dojo project:

Follow these steps to update the optimized resources currently hosted in your Dojo Project.

Before you begin

Verify that you switched your project to use the new Dojo Library project to get the Worklight Studio tooling from it. However, the project still contains all the old resources that were previously copied from the old Dojo Library into the project, which needs to be updated as well.

Procedure

1. Back up any customization you did to the www folder before you start. It is unlikely that you did any customization, but if so you must have a backup to be able to restore it later.
2. Copy the content from the `pre-built` folder of the Dojo Library project into the `www` folder of your current Dojo project.
3. Rebuild the Dojo applications in your Dojo Project to update the resources in the environments to the new ones.
4. Restart Worklight Studio to make sure that the entire tooling suite refreshes any session-specific resources.
5. Exercise your application as described in "Working with the Dojo Library Project that serves Dojo resources" on page 358.
6. Use the Worklight Studio tooling to copy the additional missing Dojo resources into your project
7. Merge the customizations you did in the first step, if applicable.

Changing the jQuery version for Worklight applications:

When you develop an application in Worklight Studio, the bundled version of jQuery might not be sufficient for development needs. This procedure provides instructions about how to use a different version of jQuery.

About this task

jQuery is bundled as a library within IBM Worklight. By default, every new application includes a main HTML file, which contains the following code that is required to use the embedded jQuery:

```
<script>window.$ = window.jQuery = WLJQ;</script>
```

To use a different version of the jQuery library, complete the following steps:

Procedure

1. Remove the `<script>window.$ = window.jQuery = WLJQ;</script>` code from the main HTML file of your application.
2. Add jQuery files to your project.
3. Add the `<script>` tag that refers to the files that you added in step 2.

Results

The updated version of jQuery will be used for all environments.

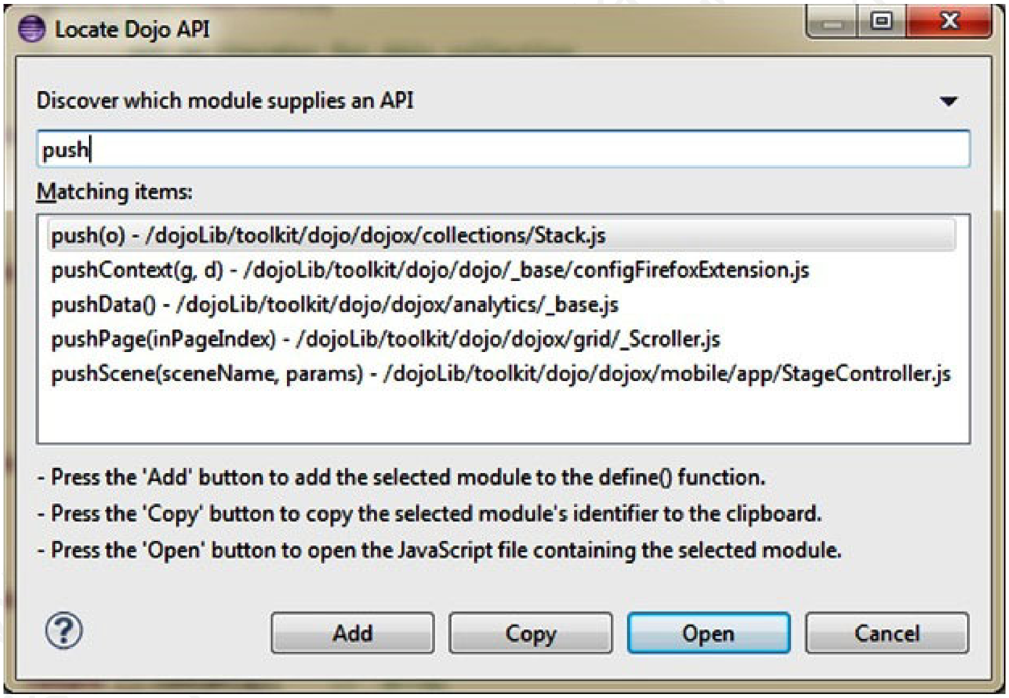
Locate Dojo API:

The Locate Dojo API dialog can be found under the **Navigate** menu and is enabled when a Dojo project resource is open in the active editor. It is enabled if a Dojo project resource is selected in a project explorer view.

The dialog locates the Asynchronous Module Definition (AMD) modules that contain the API that you need. Enter the characters of the API you need, and the locator finds the AMD modules that define that API. For example, if you were to type "push" into the search box it finds all the modules that contain types, methods, and field names that begin with "push".

Two actions are always provided once a module is selected. The **Open** action opens the JavaScript file that contains the selected module. The **Copy** action computes the selected module's path and copies it to the clipboard.

A third action that is called **Add** is provided if a JavaScript file that contains either a `require()` or `define()` function is open in the active editor. When **Add** is selected, the module's path is automatically inserted into the appropriate `require()` or `define()` function.



push

push(o) - /dojoLib/toolkit/dojo/dojox/collections/Stack.js
pushContext(g, d) - /dojoLib/toolkit/dojo/dojo/_base/configFirefoxExtension.js
pushData() - /dojoLib/toolkit/dojo/dojox/analytics/_base.js
pushPage(inPageIndex) - /dojoLib/toolkit/dojo/dojox/grid/_Scroller.js
pushScene(sceneName, params) - /dojoLib/toolkit/dojo/dojox/mobile/app/StageController.js

- Press the 'Add' button to add the selected module to the define() function.
- Press the 'Copy' button to copy the selected module's identifier to the clipboard.
- Press the 'Open' button to open the JavaScript file containing the selected module.

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.

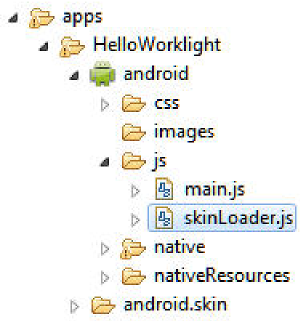


Figure 33. The `skinLoader.js` file

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 changes:

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 the following interface elements:

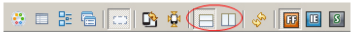
- Mobile Navigation, Outline, and Properties views
- Toolbar buttons
- Menu bar options
- Pop-up (right-click) menus
- Palette components

Note: Since IBM Worklight V6.0, the jQuery Mobile widget of Worklight Studio might be not visible in the palette of the Rich Page Editor if you are using jQuery V1.3.2. To resolve this issue, use jQuery Mobile V1.3.1 instead of jQuery Mobile V1.3.2.

Table 55. Rich Page Editor views

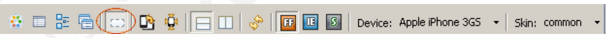
Editor view	Description
Source	The Source view helps you to view and work directly with the source code of a file. The Mobile Navigation, Palette, Outline, Page Data, and Properties views have features that supplement the Source view.

Table 55. Rich Page Editor views (continued)

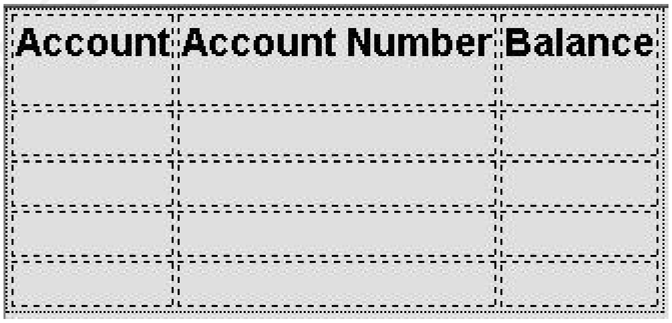
Editor view	Description
Split	The Split view combines the Source and Design views in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically. 
Design	The Design view is a WYSIWYG environment. 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: • Drag items from the Palette and Enterprise Explorer views. • Rotate the screen orientation when you use a mobile device profile to view your mobile web page in either portrait or landscape mode. • 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. • 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. • 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 to another device-specific style to view the layout and appearance of your page as it would appear on this specific device. **Note:** The **Skin** list is available only for Worklight application pages.

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.

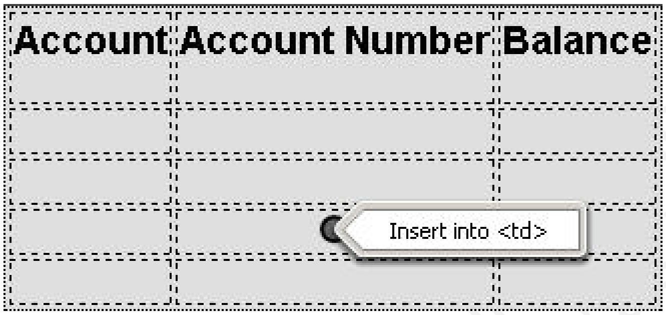


Account	Account Number	Balance

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.

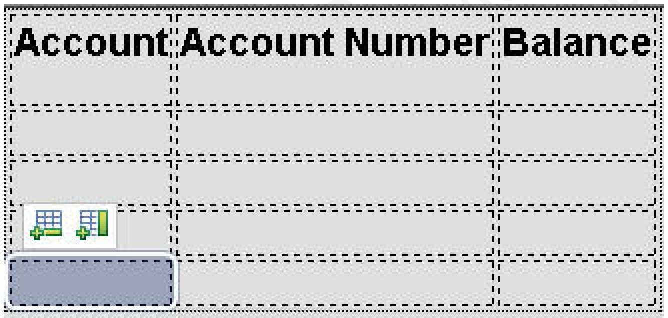
Account	Account Number	Balance

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:



Account	Account Number	Balance

Selecting a cell in a table opens a pop-up cue that you can use to add a column or row:



Account	Account Number	Balance

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.

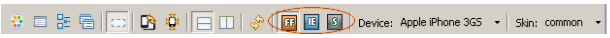
Procedure

The following table lists and describes the supported browsers in Rich Page Editor, by platform:

Platform	Supported browsers
Windows	**Internet Explorer** Available for all installations; uses the native browser code in Windows. **Firefox** 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. **Safari** Safari for Windows can be installed separately. After installation Rich Page Editor can be used in Safari. Safari support is only available on 32-bit installations of the product.
Linux	**Firefox or WebKit** The product attempts to locate and use browser code: • WebKitGTK+libraries • XULRunner installation 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."
Mac	**Safari** The native Safari browser is automatically used for products that are available on the Mac platform.

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

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 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, version mismatches 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:

   For users of IBM Worklight only:

   - 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.
   - Some IBM Worklight Studio installations use JRE arguments from the `Worklight.sh` script instead of from the `eclipse.ini` file.
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.

   Note:

   For users of IBM Worklight only: 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.

Editor preference	Description
Design mode	Specify whether to enable or disable Design Mode. 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. 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.
Web browser	Select the web browser in which to show the page that is being edited. **Note:** 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**Remember settings for each individual resource**.
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. **Tip:** By default, matched nodes are highlighted in yellow. To change the highlight color, click **Change Highlight Color**.

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 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 56. 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.
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 **Edit** 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 Ctrl + C and Ctrl + V 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: 1. From the main menu, select **Window** > **Preferences** > **General** > **Editors** > **Structured Text Editor**. 2. On the Structured Text Editor preferences page, select **Report problems as you type**.

Table 56. Source view features (continued)

Feature	Description
Customization	You can customize the appearance of the editor on either of the following preferences pages: • **Window** > **Preferences** > **General** > **Editors** > **Editors (or Structured Text Editors)** • **Window** > **Preferences** > **Web** > **HTML Files** > **Editor**

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. Right-click the editor window and select **Source** > **Format**.
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 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 project.
2. Set the target device for your 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 Dojo 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.
3. Optional: To open the New Web Page Options page and add more options to your mobile web page, click **Options**.

Option	Description
Set the document type declaration to HTML 5 and cache the page	1. From the list of options, click **Document Markup**. 2. From the **Document Type** list, select **HTML 5** to show more options. 3. Specify the icon that is used by mobile devices when users add bookmarks. To select an icon from your workspace, click **Browse** next to the **File href** field. 4. Enable browser application caching. In the **Manifest Section** field, select **CACHE** 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: http://dev.w3.org/html5/spec, and search for "cache manifest".

Option	Description
Set the device detection and stylesheet options	1. From the list of options, click **Mobile Web Page**. 2. Select one of the following options: **Detect device** The web page detects the device that shows the content and loads the appropriate CSS by including the script `dojox/mobile/deviceTheme.js`. **Select `dojox.mobile` stylesheet** The selected style sheet is loaded by using the `<link>` tag. You can select one of the following style sheets: • `blackberry.css` • `android.css` • `ipad.css` • `iphone.css` **No CSS** 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 **Stylesheets** from the list of options and add the style sheets that you want to use.

4. Click **Finish**. Your web page opens in Rich Page Editor.

Mobile patterns:

Mobile patterns provide templates that you can use to develop pages that are 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.

You can choose from many mobile patterns available in the Default Mobile Pattern Set, or you can create your own Mobile Pattern Sets. See Adding a UI pattern to a Pattern Project.

All the available Pattern Sets in your workspace and the Default Mobile Pattern Set appear grouped in the Pattern Set combination box. You can select any Mobile Pattern Set and see its content on the Add Mobile Page window.

Each Pattern Set contains categories and each category groups a list of patterns, for example: The Default Mobile Pattern Set are grouped into four categories.

Selecting a category on the Add Mobile Page window displays a list of available patterns that are 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 a User ID and password fields. Or, select a template that contains more input areas or buttons, such as **forgot password** and **register**.

Navigation and search

Choose from various navigation patterns, which include toolbars, navigation lists, or lists with searchable content.

Configuration

Choose from blank configuration pages to pages that contain 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**, the Reset password template page is also created. When you select a mobile pattern that is a set, you see all pages in the preview.

Choosing a mobile pattern adds the appropriate code into your application after which you can alter it as required.

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.

Before you begin

If the Mobile Navigation View is not shown, go to **Window** > **Show View** > **Other** > **Web** > **Mobile Navigation** to display it.

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 that are associated with the category are loaded in the view.
3. Required: Select the mobile pattern and click **Finish** to insert into your application.

Related concepts:

"Mobile patterns" on page 394
Mobile patterns provide templates that you can use to develop pages that are 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 a UI pattern to a Pattern Project:

The UI Pattern is a container for mobile patterns. Mobile patterns can be added to either a `Dojo` or `jquery` Worklight app. Users can add their own mobile patterns into the tool.

Procedure

1. Use the UI Pattern Project to create your own pattern project.
2. Over `WebContent` folder, open menu and choose UI Pattern wizard.
3. Define the name of your pattern and click **Finish** to insert into your Pattern Project.
4. A folder is added to the `WebContent` directory project structure, which contains your new pattern. The pattern resources are created in the `Dojo` and `jquery` subdirectories.

   Inside the folder, there is an HTML file named pattern. Here you can start creating your pattern.

   You can edit the `pattern.html` files, use RPE (Rich Page Editor). Click **add** to see that your new Pattern Project was added as a **New Pattern Set** in the combination box. Select the **Pattern Set** and your new UI Pattern is displayed in the browser.

Related concepts:

"Mobile patterns" on page 394
Mobile patterns provide templates that you can use to develop pages that are 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 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.

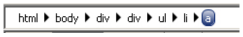
Properties view associated with Rich Page Editor:

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 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.

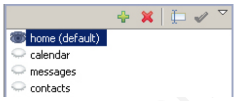
External mobile view or page
A mobile view or page that is written in a separate file or fragment. Creating mobile views or pages in separate files or fragments makes source code shorter and easier to manage.

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	You can create the following types of Dojo widgets: **View** A container that represents the entire mobile device screen. **ScrollableView** A view widget with touch scrolling capability that you can use to provide fixed position header and footer bars. **SwapView** A view widget that you can swipe horizontally to show adjacent SwapView widgets. You can create the following types of jQuery widgets: **Page** A container that represents the entire mobile device screen. **Dialog page** A container that is shown in the form of a dialog box.

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
Switch between mobile views or pages	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 **home** view is visible in Rich Page Editor; the calendar, messages, and contacts views are not visible.  To switch to the calendar view, click the icon to the left of **calendar**.
Rename mobile views or pages	Right-click the view or page that you want to rename and then click **Rename**. For example, to rename the calendar view to internet: 1. Right-click **calendar** and then click **Rename**. 2. In the **Mobile View id** field, specify `internet`.
Set the default mobile view or page	Right-click the view or page that you want to set as the default and click **Set as default**.
Remove mobile views or pages	Right-click the view or page that you want to remove and click **Remove**.

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	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. **Tip:** You can link Dojo mobile widgets to mobile views by using the **Link to Mobile View** action. 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. 2. To open the Link to Mobile View dialog, on the toolbar click **Link to Mobile View**: 3. Select one of the following options. • Click **Inline Mobile View**; from the list, select the mobile view that you want to link to the widget. • Click **Page Fragment**, and then click **Browse** to browse to the mobile page file that you want to link to the mobile view. 4. Click **Finish**.

Testing mobile applications

You can use the mobile browser simulator to emulate various mobile devices and test your mobile applications without the need to install device vendor native SDK.

Before you begin

1. Create a Worklight project.
2. Add Worklight Environments.
3. Add HTML Tags and UI widgets to your `index.html` page.

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 Eclipse select **Window** > **Preferences** > **Web Browser**. Then, select **Use external web browser**.

2. Right-click your environment folder (Android for instance) or Application folder name and select **Run As** > **Preview**.

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.

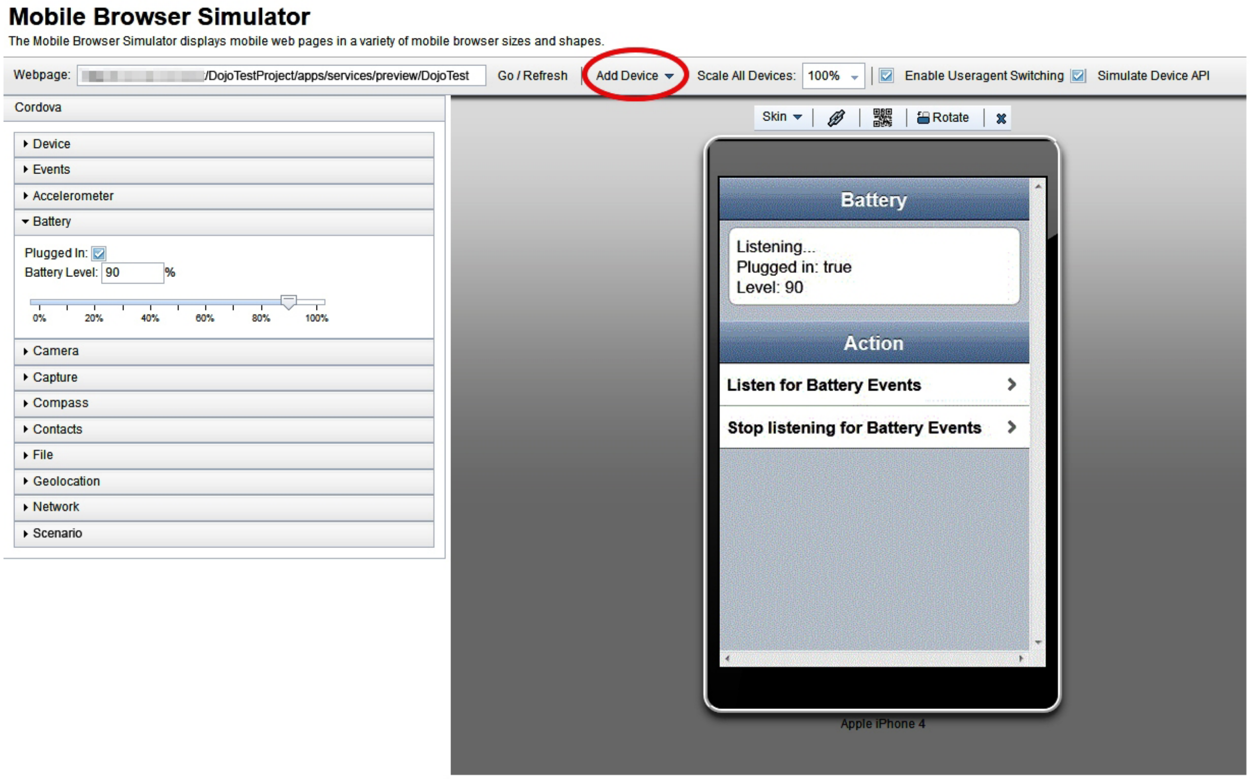
Switching devices:
Before you begin

To view your web application in the simulated devices by using the appropriate style sheets, ensure that these tasks are completed:

1. "Creating web pages for mobile devices" on page 392
2. Enable user agent switching.

Procedure

In the simulator, click the device list and then select the device that you want to simulate.



Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.

Webpage: [redacted]/DojoTestProject/apps/services/preview/DojoTest

Scale All Devices: 100%

☑ Enable Useragent Switching ☑ Simulate Device API

Plugged In: ☑
Battery Level: 90 %

Listening...
Plugged in: true
Level: 90

Adding devices:
Before you begin

To view your web application in the simulated devices by using the appropriate style sheets, ensure that these tasks are completed:

1. "Creating web pages for mobile devices" on page 392
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 start 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 8, and mobile web application environments.

Tip: When you preview a Worklight application on an Android, iPhone, iPad, BlackBerry 6 and 7, or Windows Phone 8 environment, only the devices for the created environments are available. For example, if you preview a Worklight application on an Android environment, you can select from the list of available Android devices and also the device lists from any other environments added to your application.
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.

All environments can be previewed from the application folder. Each environment-specific preview allows for the addition of devices from available environments.

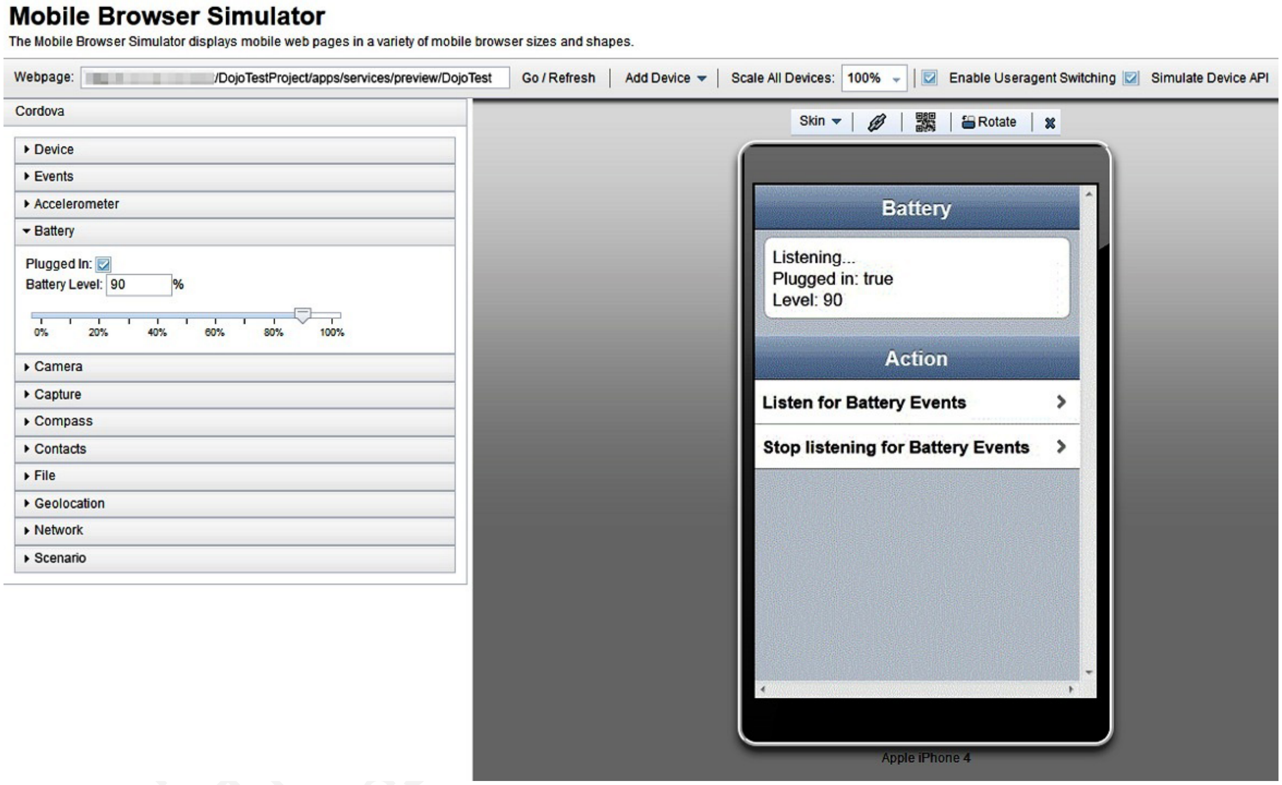
Skins can be tested per device in the mobile browser simulator. Only skins available for that platform is shown. A file can be saved in Worklight Rich Page Editor and then instantly previewed by clicking **Go/Refresh**

The link icon on the device toolbar can be selected to debug an application in a separate, simple preview.

Whenever a new environment or skin is added to a Worklight app, the mobile browser simulator must be restarted from Eclipse, **Run As** > **Preview**. Only from the Eclipse Studio **Run As** > **Preview** supports the changing of skins. The console preview does not support the changing of Worklight skins.

The Quick Response (QR) code icon on the device toolbar can be selected to show a QR code specific to the environment's URL. This QR code generator therefore allows for quick testing on a physical device.

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, 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.

Webpage: [redacted]/DojoTestProject/apps/services/preview/DojoTest

Scale All Devices: 100%

☑ Enable Useragent Switching ☑ Simulate Device API

Plugged In: ☑

Battery Level: 90 %

Listening...
Plugged in: true
Level: 90

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:

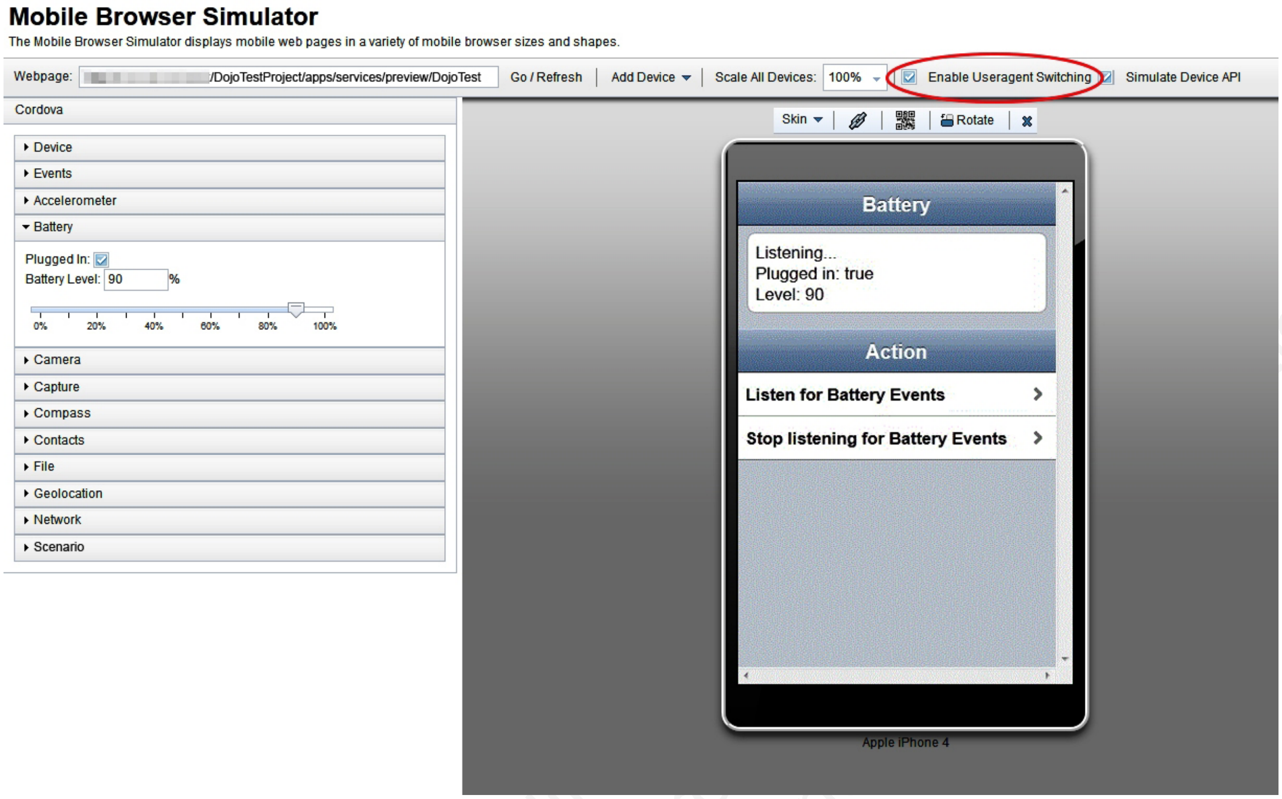
- Mozilla Firefox.
- Chrome 17 and later, with limitations.

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.

Webpage: /DojoTestProject/apps/services/preview/DojoTest

Scale All Devices: 100%

☑ Enable Useragent Switching

☑ Simulate Device API

Plugged In: ☑

Battery Level: 90 %

Battery

Listening...
Plugged in: true
Level: 90

Results

You can now view your web application with the appropriate style sheets and theme in the simulated mobile devices.

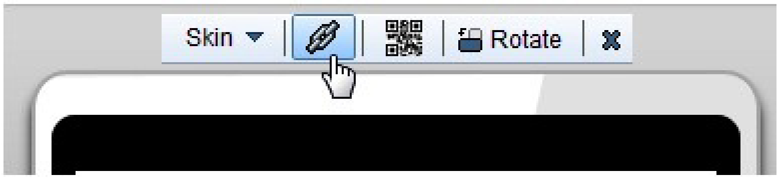
Preview web resource changes on an emulator or mobile device

During development, you can build and deploy a hybrid application to an emulator or to an actual native device to test its function.

If the web resources are still being changed frequently, some additional setup to the deployed test application can speed up the preview process between revisions. With the modified configuration, the native app can update itself to use the latest web artifacts in your Worklight Studio workspace without the need to rebuild and redeploy the application after each change.

To enable the faster preview and refresh cycle, replace the name of the application's main page with the full URL of the application that is running on the preview server. To find the correct preview URL, follow these steps:

1. Under the hybrid application's root folder in the Worklight Studio workspace, find the environment folder that you plan to test on a native device, for example Android, iPad. Right-click the folder and start **Run As** > **Preview** to open the mobile browser simulator with the page. This action starts the Worklight Development Server if necessary.
2. When the page opens in the mobile browser simulator, find and click the “link” icon in the toolbar above the preview page:



3. This action opens a new page in the browser that points directly to the specific environment's preview page. Copy the URL of the preview page, as exampled here: `http://[servername]:10080/[project name]/apps/services/preview/[app name]/[environment]/1.0/default/index.html`

Next, you need to paste this URL into the relevant configuration file within the native application resources:

1. To begin, select the application's folder in the navigator and perform **Run As** > **Build Only (All Environments)** . This action builds the native resources from the current project source.
2. Underneath the environment folder that you plan to test natively, find the “native” folder. The configuration file is located under this folder.
   - For Android environments, edit `/native/assets/wlclient.properties`
   - For iPhone and iPad environments, edit `/native/worklight.plist`
3. Find the value of the `wlMainFilePath` or `wlMainFile` configuration parameter (whichever is present). The default page name is `index.html`.
4. Replace the page name with the full URL you previously copied from the mobile browser simulator page.

Note: As soon as you change these files, a prompt appears to change the file from a read-only state. Press Yes to commit your changes, then save the file.

After you save the configuration file, start the application on the native emulator or mobile device by using the normal process for the environment you are previewing.

As you continue to develop your web resources, you can update the native application in one of two ways:

- If the device or emulator supports accelerometer events, shake the device vigorously for a short time until a prompt dialog appears. Click **Refresh** in the dialog to update to the latest web resources.
- For scenarios where shake-detection is unavailable, simply close the application and relaunch it within the emulator or native device. When the application restarts, it retrieves the latest web resources from the Worklight Development Server.

  Note: The shake preview feature is for web resource preview, and does not use the native device features.

  Note: The Worklight Development Server must be running for the application to function correctly under this modified configuration. An error message indicates whether the application cannot connect to the preview server.

Note: The **Build Only (All Environments)** action overwrites the configuration files that you changed in the previous process. Generally you can continue to develop and preview the web resources in your hybrid application without doing any rebuilds, and then run the build action again after application development is complete to generate the final native applications. If it is necessary to rebuild frequently, optionally re-execute the previous steps to restore the faster preview function.

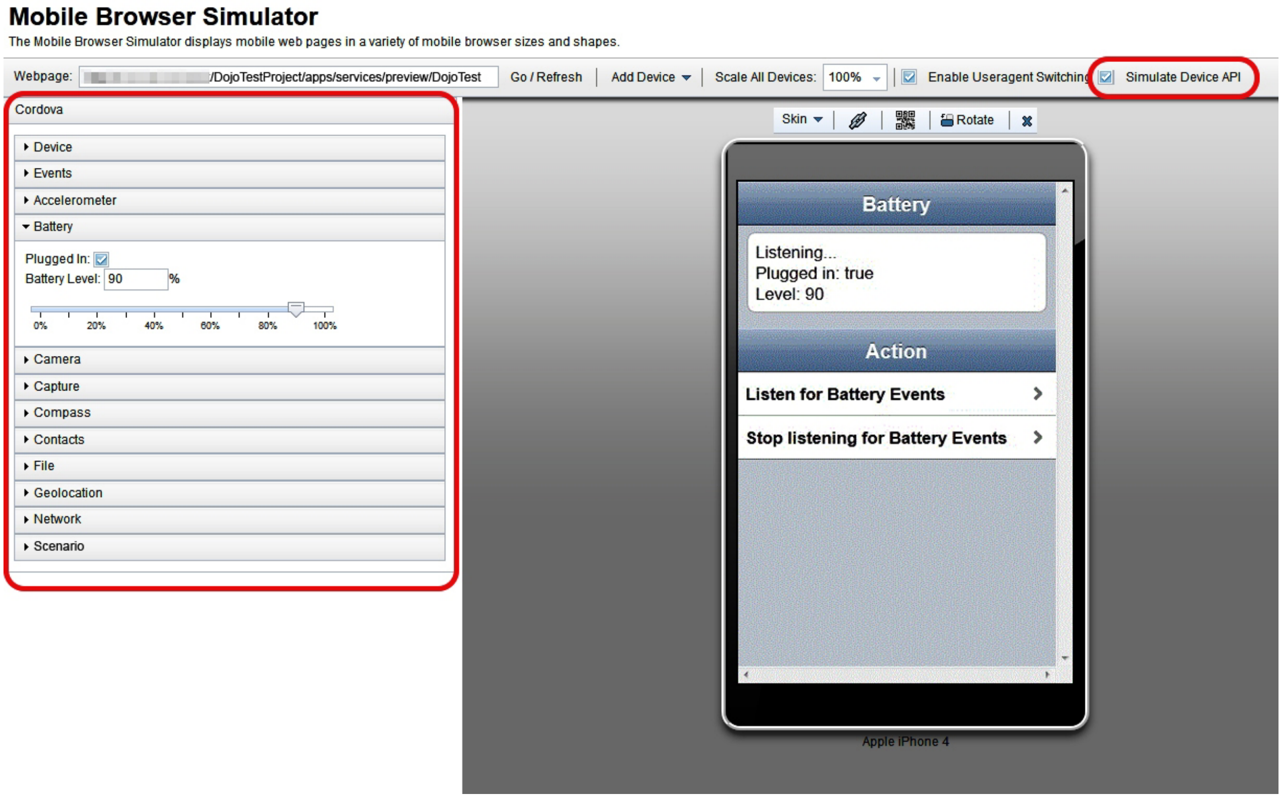
Previewing your Worklight applications

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 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 8
- Windows 8 desktop and tablets



Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.

Webpage: /DojoTestProject/apps/services/preview/DojoTest

Scale All Devices: 100%

☑ Enable Useragent Switching ☑ Simulate Device API

Battery

Plugged In: ☑

Battery Level: 90 %

Listening...

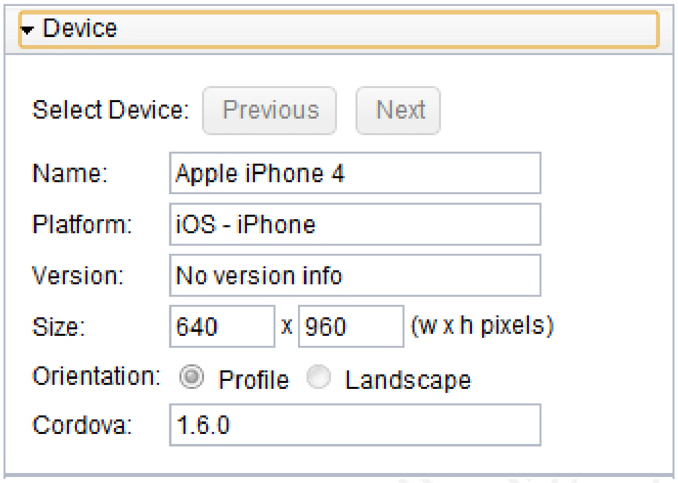
Plugged in: true

Level: 90

Tip: If you do not want to use the Cordova simulation to preview your Worklight applications, clear **Cordova** 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

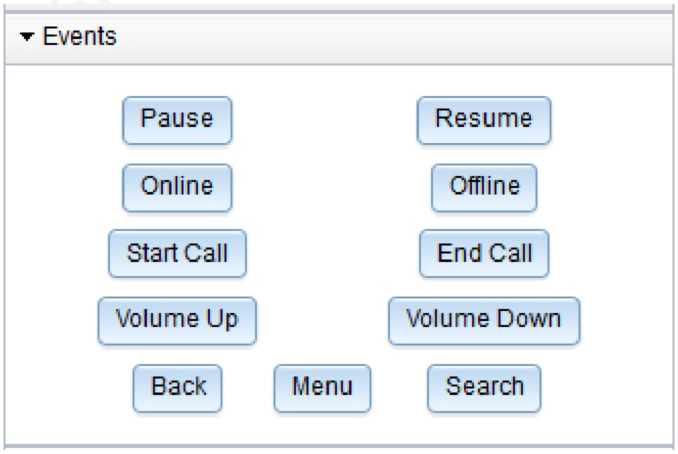
Cordova: 1.6.0

Events

Triggers any of the following Cordova events:

- pause
- resume
- online
- offline
- backbutton
- menubutton
- searchbutton
- startcallbutton
- endcallbutton
- volumedownbutton
- volumeupbutton

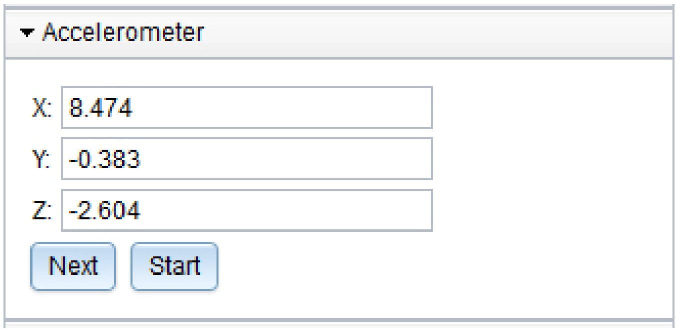
To trigger a Cordova event, click the corresponding button:



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: 8.474

Y: -0.383

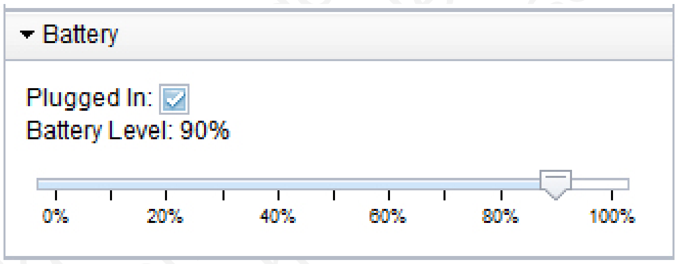
Z: -2.604

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 **Plugged In**.


Battery

Plugged In: ☑

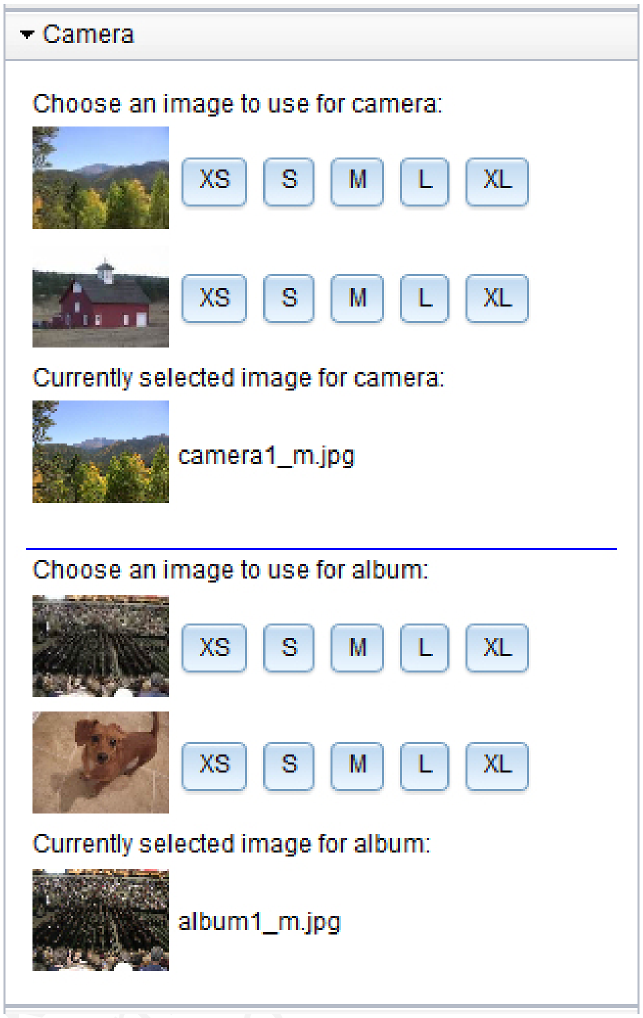
Battery Level: 90%

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**.



Choose an image to use for camera:

Currently selected image for camera: camera1_m.jpg

Choose an image to use for album:

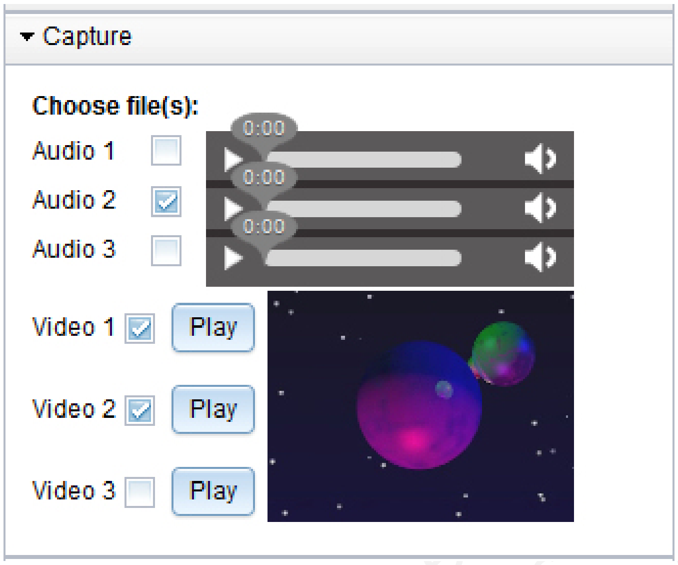
Currently selected image for album: album1_m.jpg

Capture

Simulates the Cordova capture API by using the following methods:

- capture.captureAudio
- capture.captureVideo

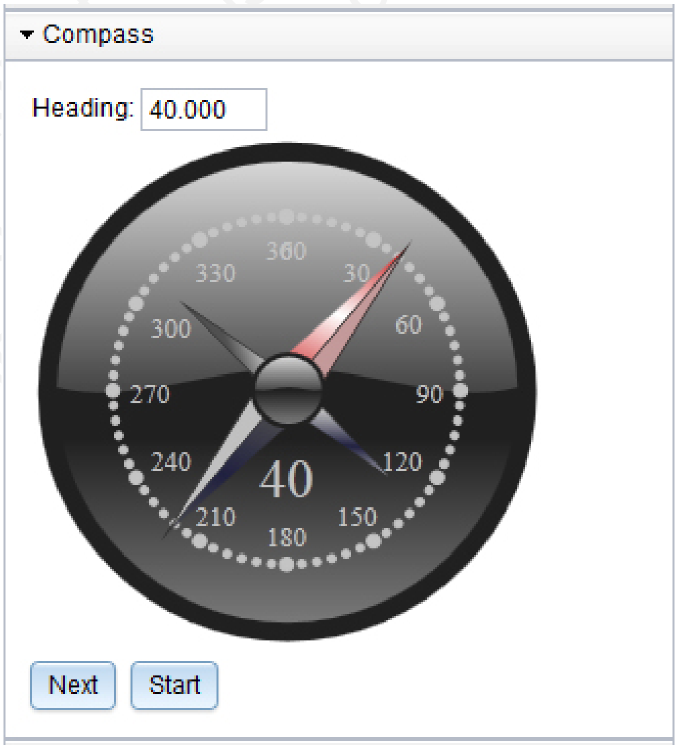
You can select the audio and video recordings that you want to use, and play these recordings by using the HTML5 players.



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.

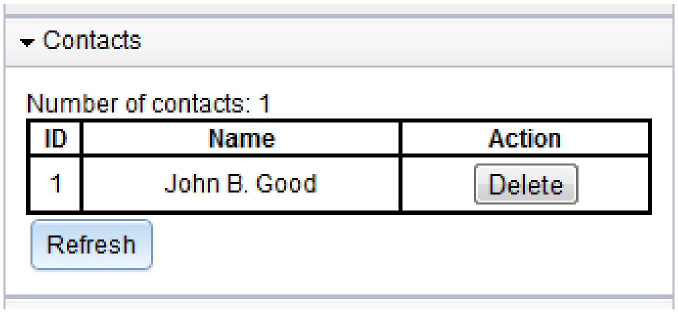
Compass

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.


Heading: 40.000

Contacts

Shows the available contacts for the mobile device. You can delete contacts and refresh the list of available contacts.

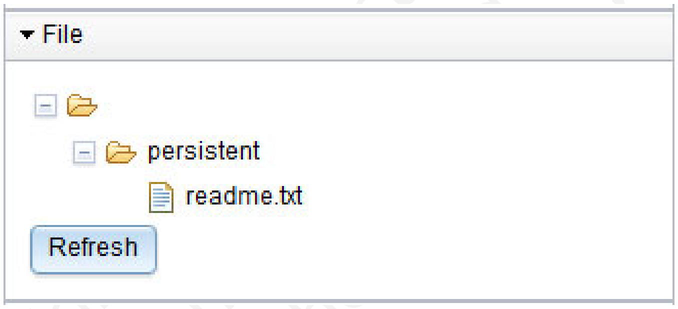


Number of contacts: 1

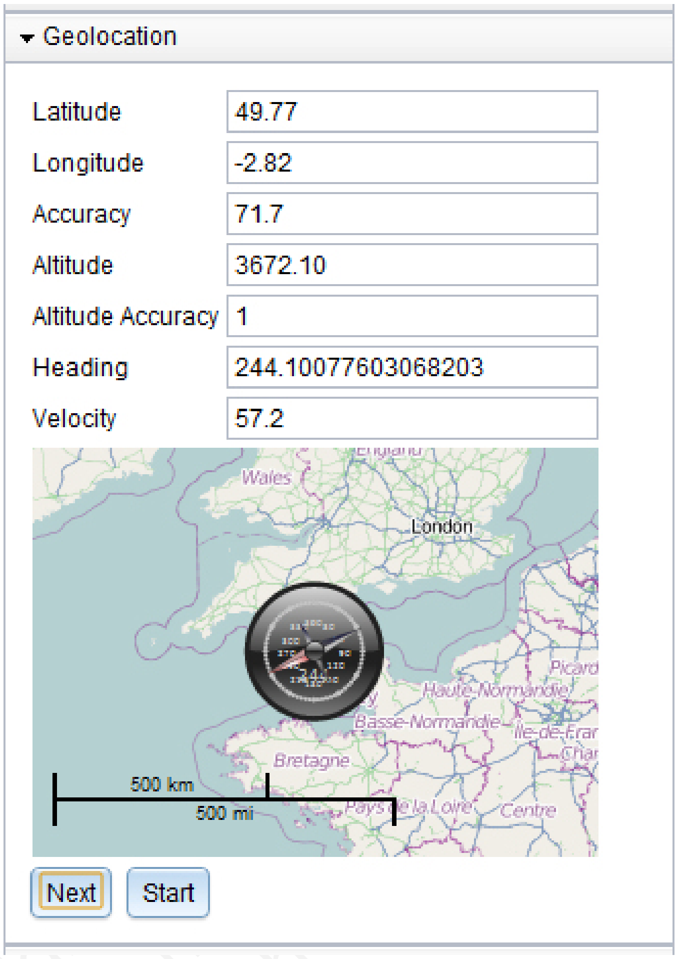
ID	Name	Action
1	John B. Good	Delete

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.



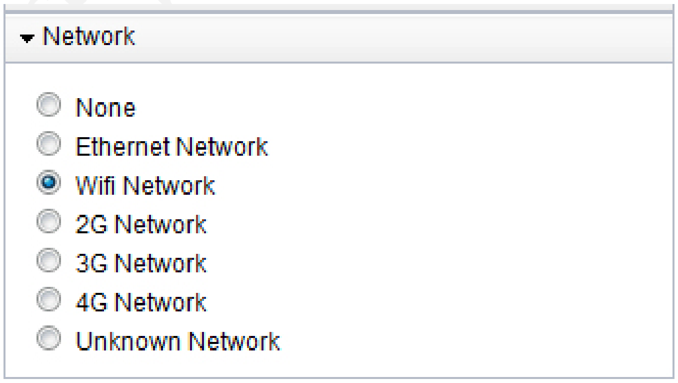
Geolocation
Generates the Geolocation values returned by the Cordova API when querying Geolocation data. To generate a new set of values, click **Next**. To generate the values periodically, click **Start**.



Geolocation

Latitude: 49.77
Longitude: -2.82
Accuracy: 71.7
Altitude: 3672.10
Altitude Accuracy: 1
Heading: 244.10077603068203
Velocity: 57.2

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 the IBM Worklight client API

IBM Worklight defines a series of client APIs that you can use in your apps.

You can use the client APIs to 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, 7, and 10, and Windows Phone 8 devices
- Work offline
- Use encrypted cache

For more information about these client APIs, see "IBM Worklight client-side API" on page 695.

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 WL.Client.connect method, as defined in the WL.Client class. 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
});
```

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'), undefin
 //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: {leve
```

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.
```

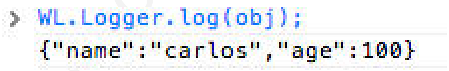
Stringify

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.

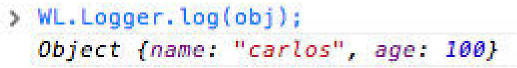
Turn on the stringify function:

```
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 invokeProcedure method as defined in the WL.Client class.

```
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 the logActivity method as defined in the WL.Client class.

```
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`.

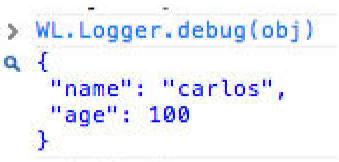
```
WL.Logger.off().on().ctx({pkg: 'com.my.app'}).log('Hello')
//'[com.my.app] Hello'
```

Pretty-print JSON objects

You can format JSON objects by enabling `stringify`.

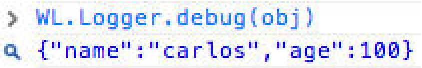
By enabling "Stringify" on page 418 (`stringify: true`) you can display JSON objects in a more readable format.

```
var obj = {name: 'carlos', age: 100};
WL.Logger.on({stringify: true, pretty: true});
```



```
> WL.Logger.debug(obj)
{
  "name": "carlos",
  "age": 100
}
```

```
WL.Logger.on({pretty: false});
```



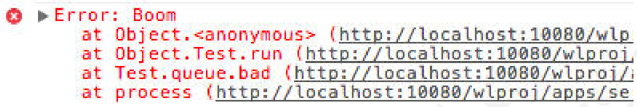
```
> WL.Logger.debug(obj)
{"name":"carlos","age":100}
```

Print stack traces

You can print stack traces for certain objects.

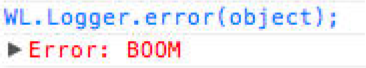
You can print stack traces for an object if the object is an instance of Error (if (object instance of Error) evaluates true) and "Stringify" on page 418 is enabled (stringify: true).

```
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

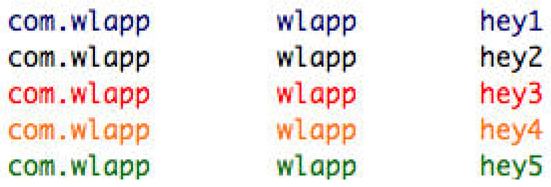
Provides better output in LogCat

Requires that the deviceready event previously fired.

Console log

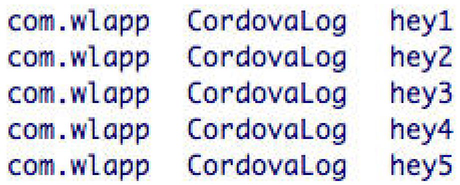
Synchronous

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 421 “Callback” on page 419,“Log message tags” on page 420, “Log package whitelist and blacklist” on page 417, and “Select log levels” on page 415.

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 V6.0. 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 1for 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

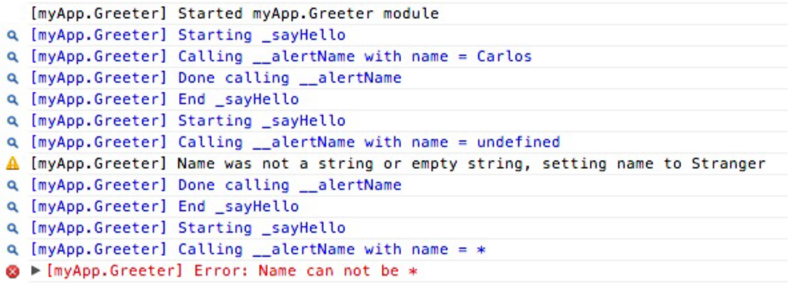
```
var myApp = myApp || {};
myApp.Greeter = (function (WL) {

    //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
```

main.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
}
```

index.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/main.js"></script>

    <!-- ... other script tags -->
</body>
```



```
[myApp.Greeter] Started myApp.Greeter module
[myApp.Greeter] Starting _sayHello
[myApp.Greeter] Calling __alertName with name = Carlos
[myApp.Greeter] Done calling __alertName
[myApp.Greeter] End _sayHello
[myApp.Greeter] Starting _sayHello
[myApp.Greeter] Calling __alertName with name = undefined
[myApp.Greeter] Name was not a string or empty string, setting name to Stranger
[myApp.Greeter] Done calling __alertName
[myApp.Greeter] End _sayHello
[myApp.Greeter] Starting _sayHello
[myApp.Greeter] Calling __alertName with name = *
▶ [myApp.Greeter] Error: Name can not be *
```

Figure 34. Log output

Developing hybrid applications with IBM Worklight Application Framework

This collection of topics describes the various stages of application development when using IBM Worklight Application Framework (Beta code).

Overview of IBM Worklight Application Framework

By providing a single editor, the IBM Worklight Application Framework helps you create hybrid applications that are designed to interact with back-end services. When you choose to build an application with IBM Worklight Application Framework, the IBM Worklight Application Framework library and the Dojo library are automatically loaded into your workspace. Those libraries are then used by the application at run time.

The IBM Worklight Application Framework editor is a multi-tabbed editor which helps you configure interactions between services, data objects, and views.

In the context of IBM Worklight Application Framework, a service is a remote information source from which you can retrieve data when you invoke the service. IBM Worklight Application Framework supports SOAP-based web services or services from the SAP NetWeaver Gateway.

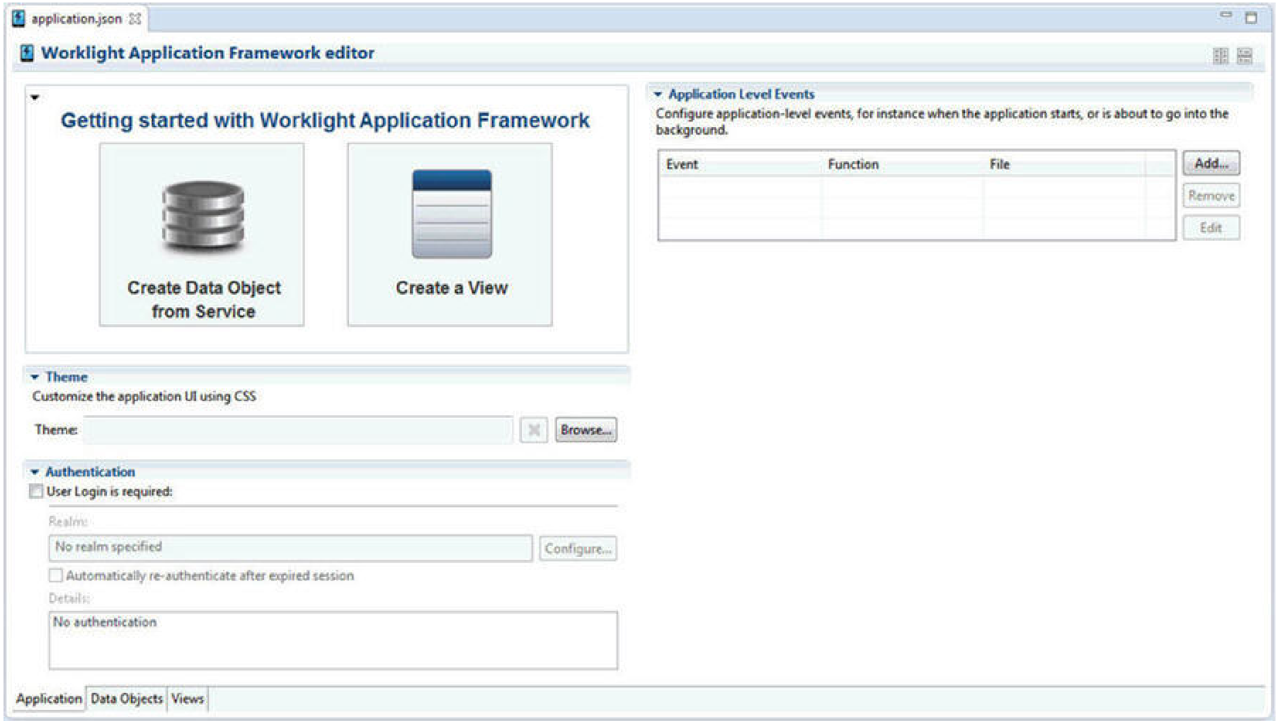
The data that you access from a specified service is composed of data objects. You can choose to execute various operations (such as Create, Retrieve, Update, Delete, or Query) on data objects. Operations are connection types that are used to model the relation between a data object and a service. For example, you can choose to retrieve data from a service, or to update it.

Data objects contain key attributes that are used within the UI of your application. You can configure the different views in which these attributes are displayed. A view defines how the data in your app is presented to the user. For example, a view can be a screen that displays the contact details of a customer.

Workflow of the development process

When you start the wizard to create an IBM Worklight application, you choose to create a hybrid app, and select **Use Worklight Application Framework (beta)**. When the application creation wizard closes, the **Application** tab of the IBM Worklight Application Framework editor opens (instead of the index.html file for other hybrid apps). You use this editor to edit the `application.json` and the `view.html` files of your app, which are automatically added to your application folder.

You use the **Application** tab to specify a set of general configurations for your app.



Configure application-level events, for instance when the application starts, or is about to go into the background.

Customize the application UI using CSS

Theme:

□ User Login is required:

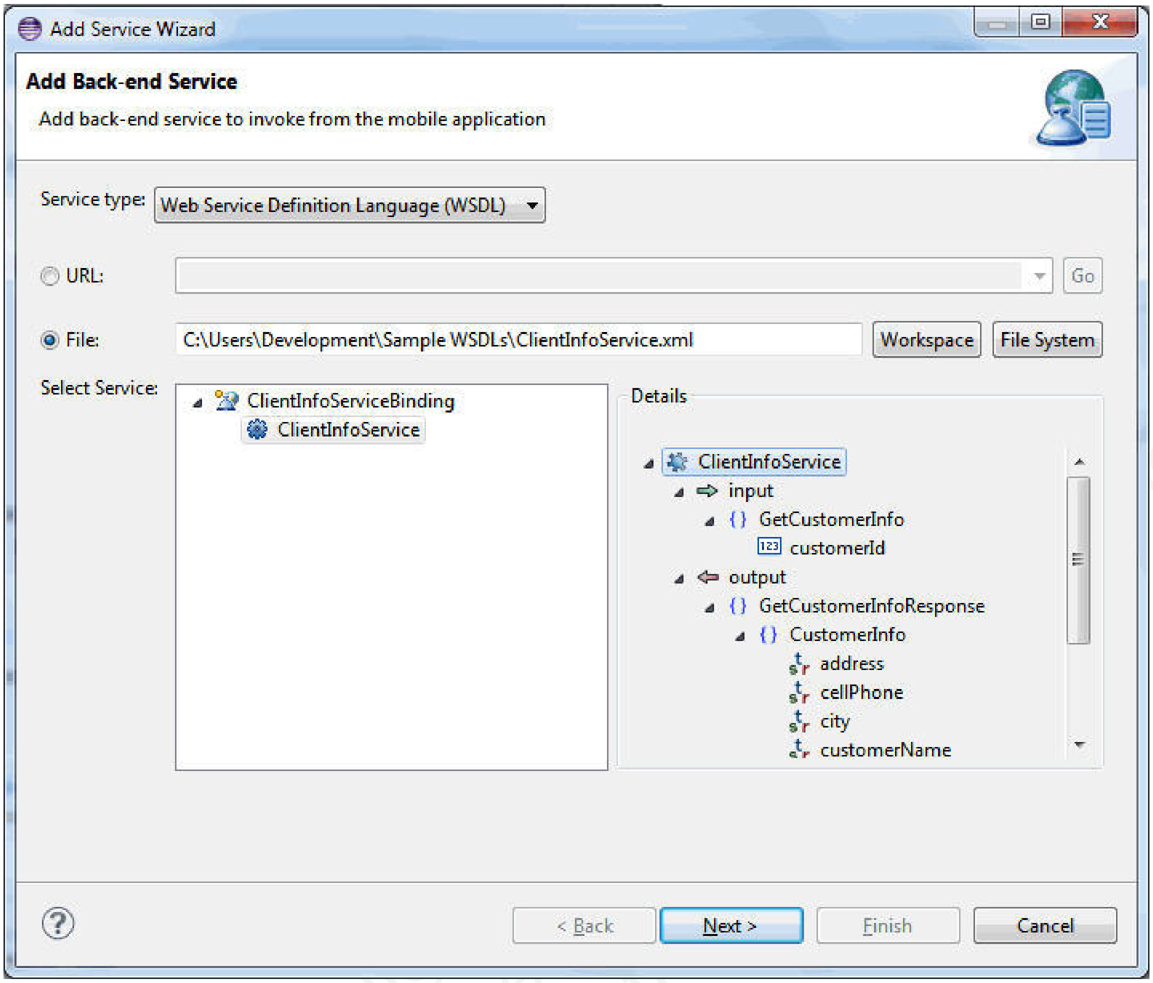
Realm: No realm specified

□ Automatically re-authenticate after expired session

Details: No authentication

Figure 35. The **Application** *tab*

In a first stage, you start the services discovery wizard to create back-end service representations, select data objects and their attributes, and choose the operations that are required by the run time.



Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: Web Service Definition Language (WSDL)

○ URL:

● File: C:\Users\Development\Sample WSDLs\ClientInfoService.xml

Select Service:

- ClientInfoServiceBinding
  - ClientInfoService

Details

- ClientInfoService
  - input
    - GetCustomerInfo
      - customerId
  - output
    - GetCustomerInfoResponse
      - CustomerInfo
        - address
        - cellPhone
        - city
        - customerName

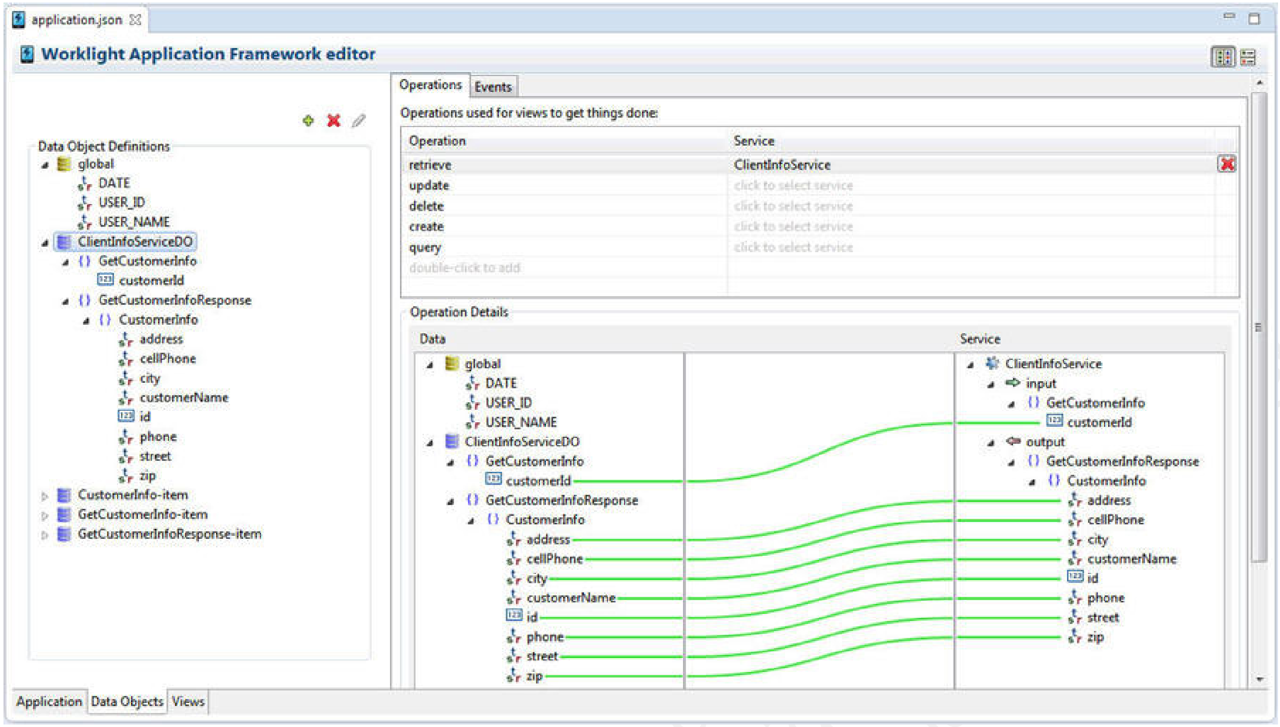
Figure 36. The services discovery wizard

Note: Alternatively, you can also choose to add services to your project and define data objects in two separate processes:

- You can add back-end services and generate the related adapters for any IBM Worklight app by following the procedure described in "Generating adapters with the services discovery wizard" on page 547.
- Then, you can define data objects for apps that are created with IBM Worklight Application Framework from the **Data Object** tab of the editor.

The **Data Object** tab then generates data object definitions based on the service interface, and mappings between the data object attributes and the service parameters for a selected operation. You can further configure those definitions and mappings in the **Data Object** tab of the editor, and customize how your app interacts with those elements.

You can also set event handlers for different levels of your application. For example, you can configure event handlers for each operation by specifying which JavaScript function to execute on events like `onBeforeInvocation` or `on AfterInvocation`.



Operations used for views to get things done:

Operation	Service
retrieve	ClientInfoService
update	click to select service
delete	click to select service
create	click to select service
query	click to select service
double-click to add	

Figure 37. The **Data Objects** tab

You then go to the **Views** tab of the editor, and create the different UI views based on the data binding that you specified, and define some transitions between those views.

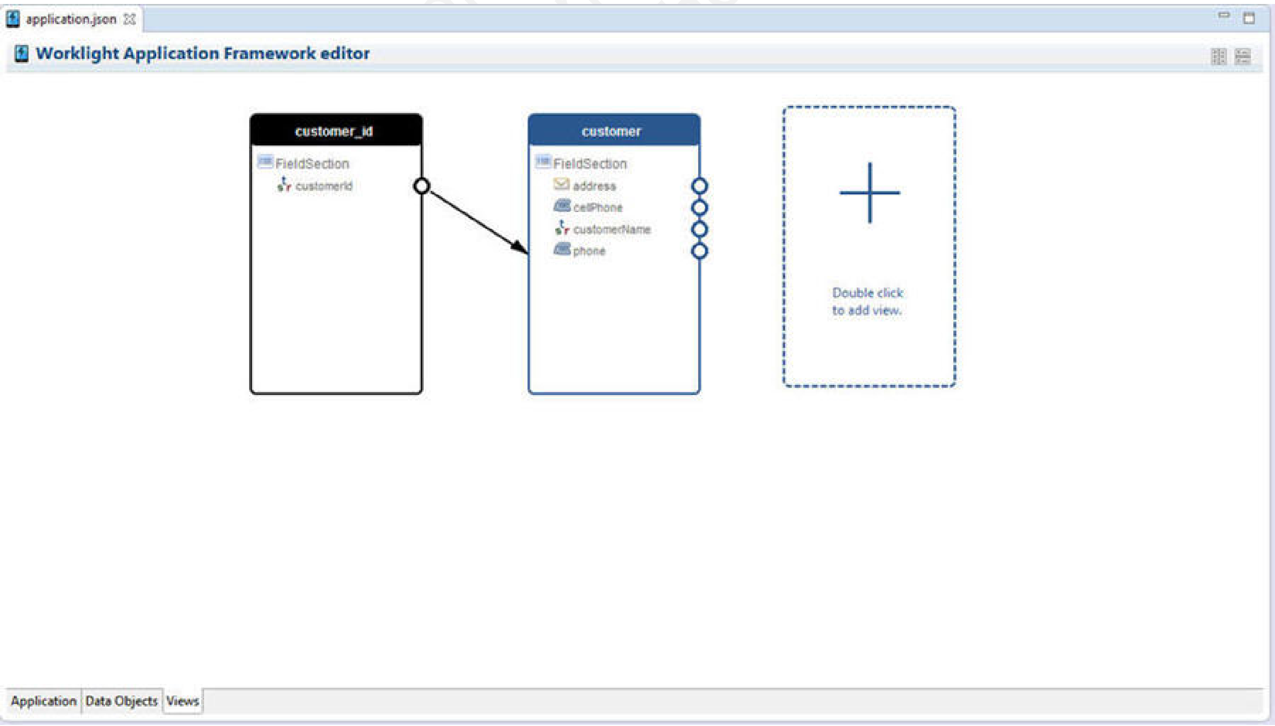
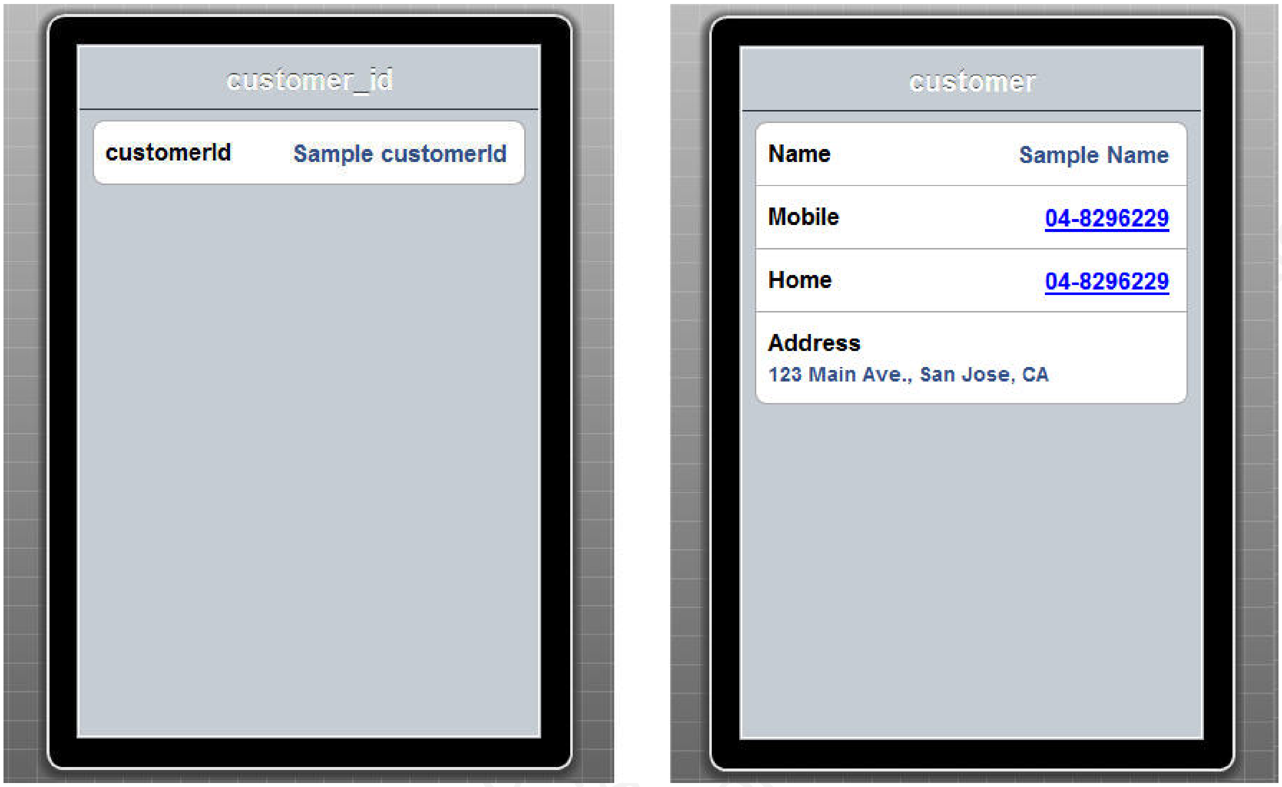


Figure 38. The **Views** tabs

To review the general structure of the generated views, reorganize fields, and rename their labels, you can edit the `views.html` file in Rich Page Editor.



customer_id

customerId Sample customerId

customer

Name Sample Name

Mobile 04-8296229

Home 04-8296229

Address

123 Main Ave., San Jose, CA

Figure 39. Preview of the new views in Rich Page Editor

The IBM Worklight Application Framework editor

Use the editor of IBM Worklight Application Framework (Beta code) in Worklight Studio to manage the interactions of your app with back-end systems and data sources, and to define how data is displayed through the UI views of your application.

General structure

The multi-tabbed editor of IBM Worklight Application Framework comprises:

- An **Application** tab: for the general configuration of your application.
- A **Data Objects** tab: to define various data objects.
- A **Views** tab: to define the different views of your application and their transitions.

You use the IBM Worklight Application Framework editor to edit the `application.json` and the `view.html` files that are used at run time.

The `application.json` file contains:

- The definitions of the data objects included in the application
- The configuration of the data that is transmitted between the data objects and the back-end services
- The connections between views and data objects
- The definition of the transitions that are made between views

The `view.html` file contains the definitions of the various views of the applications.

These files are located under the **common** folder of your application.

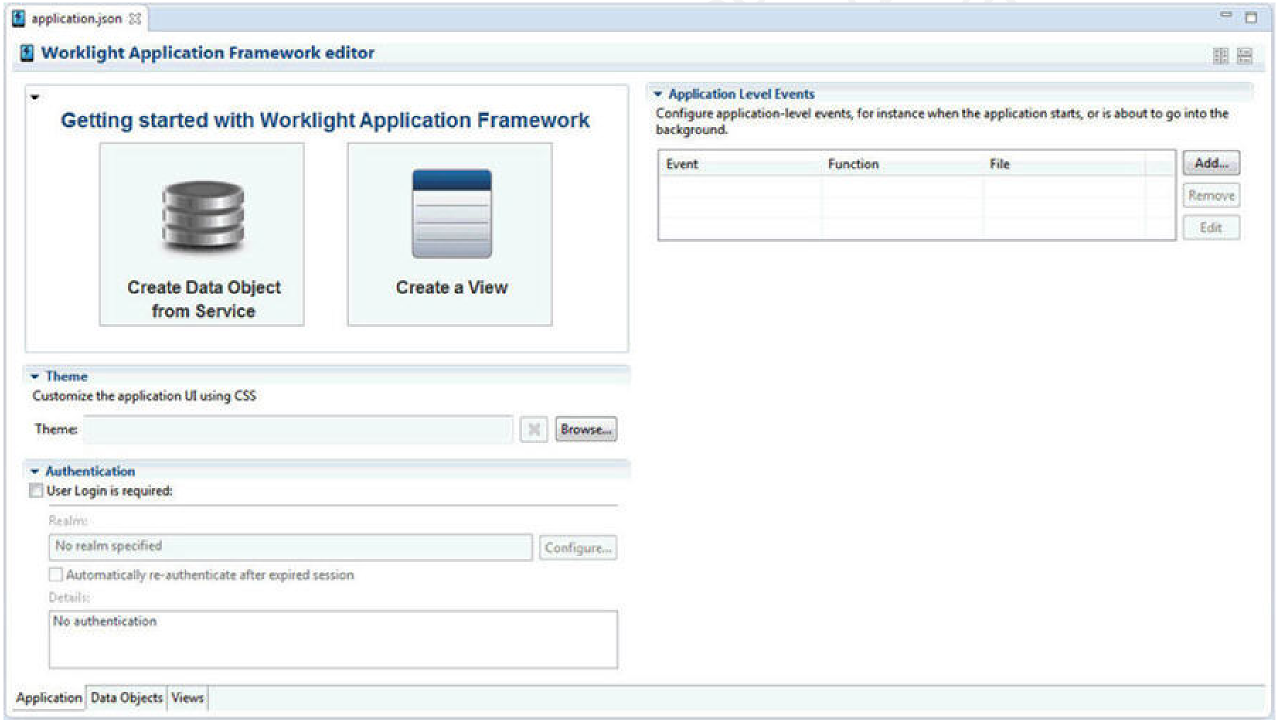
The Application tab of the IBM Worklight Application Framework editor:

Go to the **Application** tab of the IBM Worklight Application Framework editor (Beta code) to start developing your application.

Content of the Application tab

When the project creation wizard closes after you selected **Use Worklight Application Framework (beta)**, the `application.json` file automatically opens with the default IBM Worklight Application Framework editor, and displays the **Application** tab.

You can start the services discovery wizard from the **Application** tab, and add services and data objects to your app. In this tab, you can also specify a set of general configurations, such as an extra theme for your app, security parameters, and event handlers.



application.json

Worklight Application Framework editor

Getting started with Worklight Application Framework

Application Level Events

Configure application-level events, for instance when the application starts, or is about to go into the background.

Event	Function	File

Theme

Customize the application UI using CSS

Theme:

Authentication

□ User Login is required:

Realm:

No realm specified

□ Automatically re-authenticate after expired session

Details:

No authentication

Figure 40. The **Application** *tab of the IBM Worklight Application Framework editor*

Table 57. Sections of the **Application** *tab*

Sections	Description
Getting Started with Worklight Application Framework	• If you click **Create Data Object from Service**, the services discovery wizard opens. With this wizard, you define the back-end services that you want to use in your application, and the data operations that are required by IBM Worklight Application Framework at run time. For more information about the services discovery wizard, see the topic "Adding services and data objects with the services discovery wizard" on page 432. • If you click **Create a View**, an "Add View" wizard opens, and you can create a view for your application. To know how to use this wizard, see "Adding a view to your application" on page 443.
Basic Settings	You configure the general appearance of your application by adding an extra theme that supplements the `main.css` file of your app. **Note:** The extra `.css` file that you define in the **Theme** field overwrites the `main.css` file of your app only in case of conflicting properties.
Authentication	You set the security parameters of your application. First, you specify whether your application requires login. If you select **Use IBM Worklight Application Framework authentication**, you can choose the login parameters based on the realms that are available, depending on your Worklight Server configuration. For more information about realms and authentication, see "Authentication realms" on page 607.
Application Level Events	You configure custom event handlers by binding events to JavaScript functions. To specify an event and a function from your JavaScript code: 1. Click **Add**, select the event, and choose the `.js` file 2. Type the function, or click **Browse** to select a function from a list of the functions present within the chosen `.js` file.

Adding services and data objects with the services discovery wizard:

You use the services discovery wizard to specify the back-end services that you want to invoke from your IBM Worklight project. You also generate data objects that are used by the app created with IBM Worklight Application Framework (Beta code).

About this task

The services discovery wizard supports the following types of services:

- Web Services, as described by Web Services Description Language (WSDL) files. These services are procedural in nature, with inputs and outputs that are explicit. For example, when a web service calls a remote procedure, it gets a result.
- Services that are exposed by an SAP Netweaver Gateway. These services are resource-based, which means that they expose a collection of resources that you can manipulate. Like web services, they can also have custom procedural operations, and generate inputs and outputs.

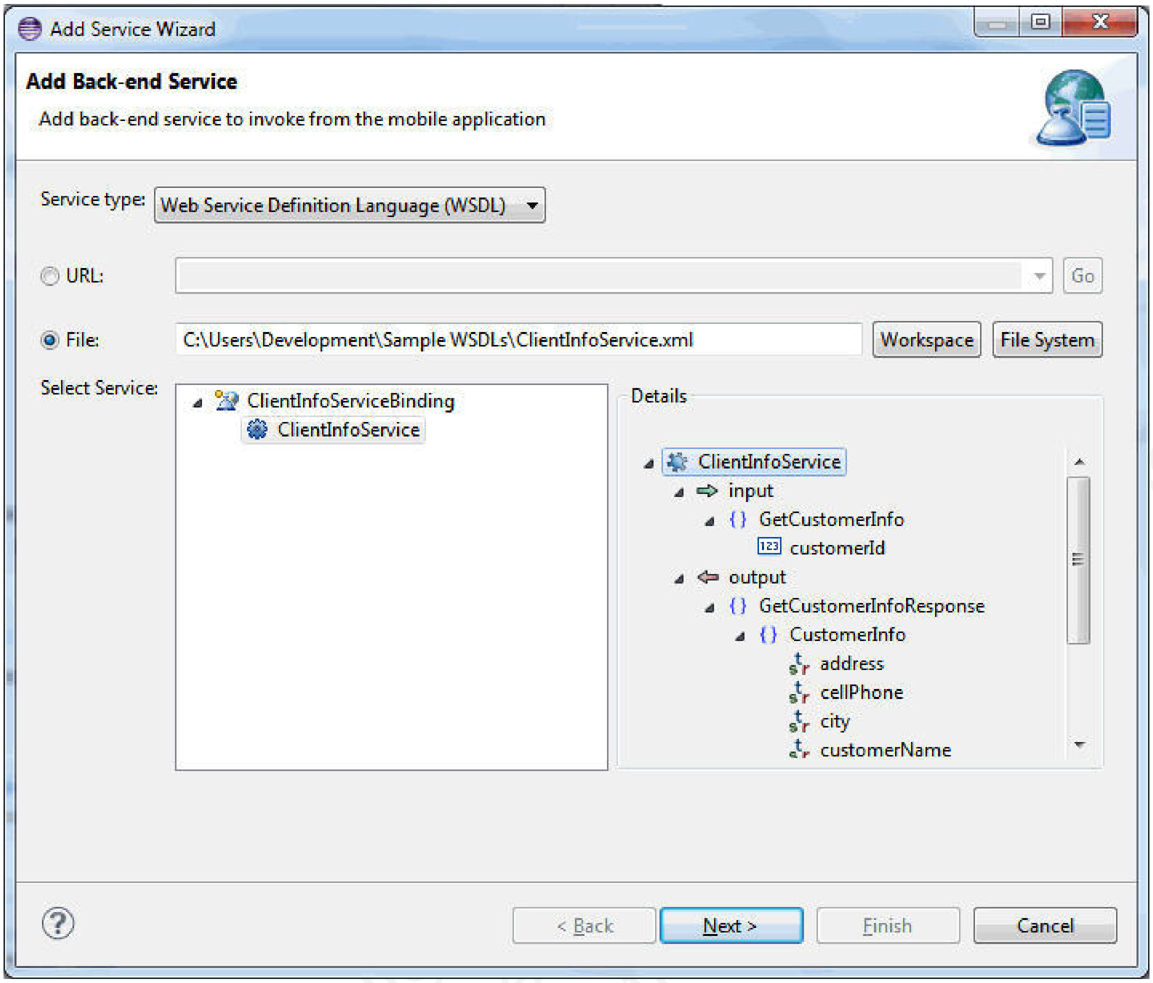
To create data objects associated with the service discovered, you specify which type of operations you want to implement on the data, such as Create, Read, Update, Delete, or Query.

The adapters that communicate with the chosen service are automatically generated, and placed in the **adapters** folder of your project.

Note: If you manually modify an adapter file, first create a copy of this file, and make sure to modify only the copied file. The services discovery wizard might regenerate the original file each time you add a service. The exact adapter that is regenerated depends on the type of service that is involved.

Procedure

1. Double-click the `application.json` file of your app to open the IBM Worklight Application Framework editor, and click **Create Data Object from Service** in the **Application** tab.
2. Select the type of service that you want to invoke from your application.
3. Depending on the selected type, define the service that you want to use, as described in the following sections:
   - **WSDL** service type:



Add Service Wizard

Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: Web Service Definition Language (WSDL)

○ URL:

● File: C:\Users\Development\Sample WSDLs\ClientInfoService.xml

Select Service:

- ClientInfoServiceBinding
  - ClientInfoService

Details

```
ClientInfoService
  input
    GetCustomerInfo
      customerId
  output
    GetCustomerInfoResponse
      CustomerInfo
        address
        cellPhone
        city
        customerName
```

Figure 41. Adding a web service

a. Enter a URL or select one from the **URL** drop-down list, and click **Go**; or browse to a file in your workspace or in your system.

Note: If you enter a secure URL (https), the system fetches the certificate from the specified server, and stores it into a private key storage area that is created in your workspace.

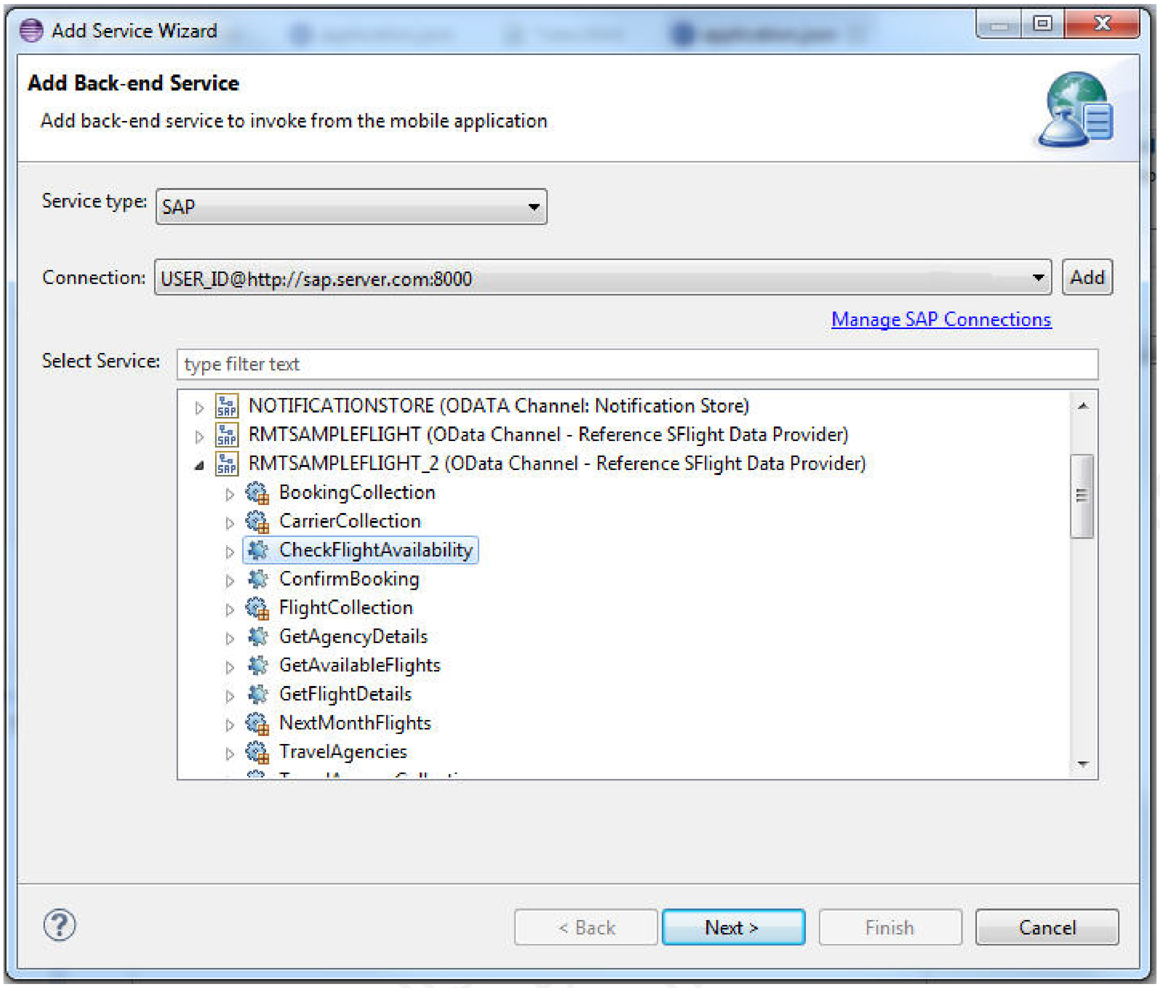
b. Optional. If you are prompted to, enter your credentials.

You can now see the list of available services. Different types of information are displayed in the **Details** pane, depending on the level you select:

- The first level corresponds to the binding configuration details. When this level is selected, the **Details** pane shows the SOAP version.
- The second level corresponds to the data operation details. When this level is selected, the **Details** pane shows the input and the output of the remote invoked procedure.

c. Select the data operation that you want to enable.

- **SAP Netweaver Gateway Services** service type:



Add Service Wizard

Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: SAP

Connection: USER_ID@http://sap.server.com:8000

Select Service: type filter text

- NOTIFICATIONSTORE (ODATA Channel: Notification Store)
- RMTSAMPLEFLIGHT (OData Channel - Reference SFlight Data Provider)
- RMTSAMPLEFLIGHT_2 (OData Channel - Reference SFlight Data Provider)
  - BookingCollection
  - CarrierCollection
  - CheckFlightAvailability
  - ConfirmBooking
  - FlightCollection
  - GetAgencyDetails
  - GetAvailableFlights
  - GetFlightDetails
  - NextMonthFlights
  - TravelAgencies

Figure 42. Adding a service exposed by SAP

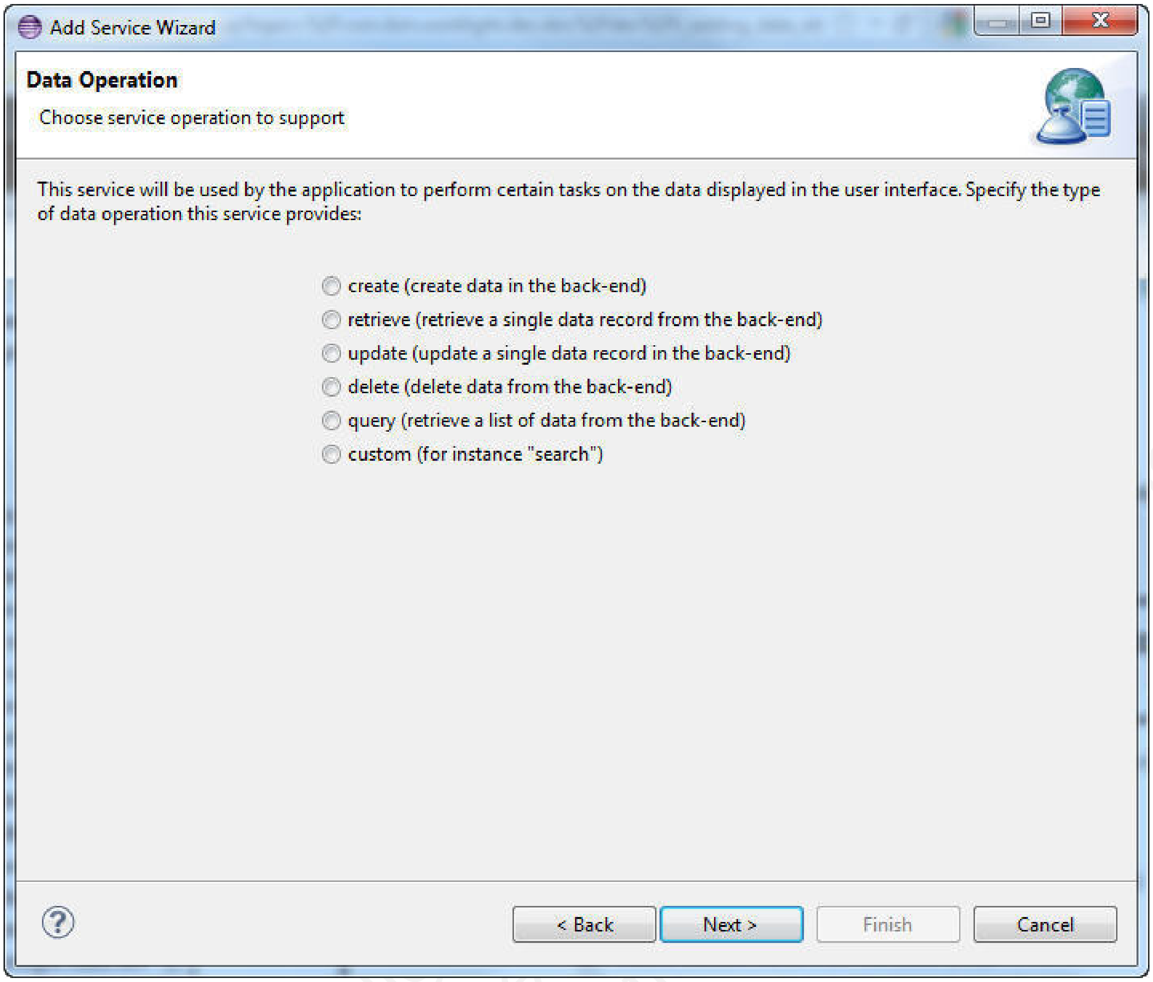
a. Set up a connection to an SAP Netweaver Gateway server by either:
   - Clicking **Add** to create an SAP connection.
   - Clicking the **Manage SAP Connections** link to edit existing connections.
   - Selecting an existing SAP connection from the **Connection** drop-down list.

b. Proceed with the connection configuration by entering your server URL, client ID, user name, and password.

   You can now see the list of SAP services that are available on the server you specified in the **Select Service** pane.

c. Expand the services nodes to select the collection or procedure that you want to use in your project. You can further expand the nodes to see what fields and operations are available for each service.

4. Click **Next**.
5. On the page that lists the supported operations, choose the type of operation available with the service you specified.
   - For procedural types of services, such as SOAP, you can define only one operation for each service discovered.



Data Operation

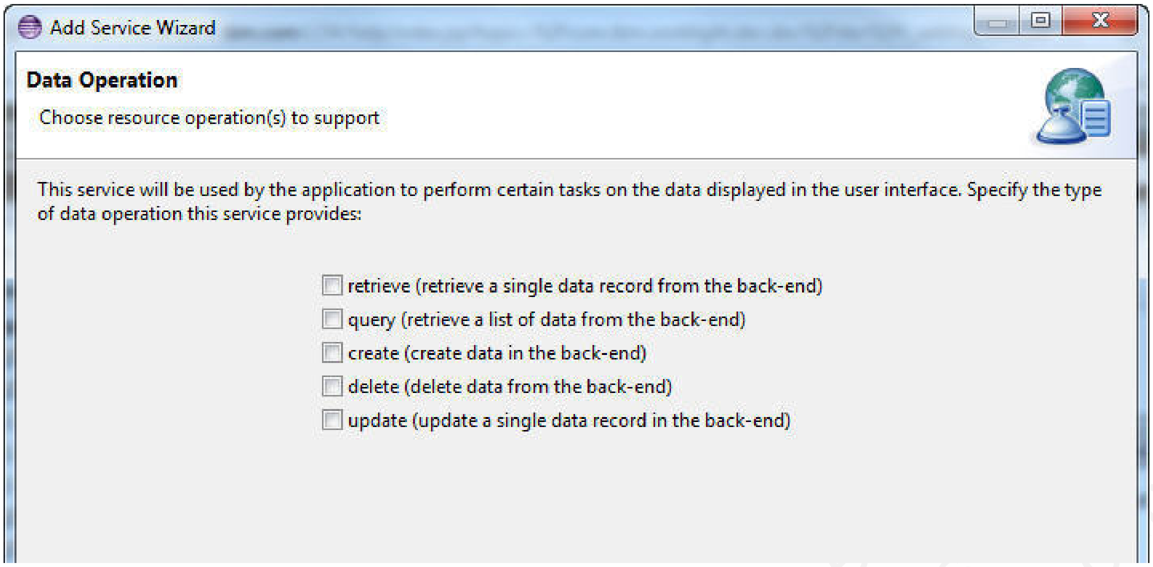
Choose service operation to support

This service will be used by the application to perform certain tasks on the data displayed in the user interface. Specify the type of data operation this service provides:

- ○ create (create data in the back-end)
- ○ retrieve (retrieve a single data record from the back-end)
- ○ update (update a single data record in the back-end)
- ○ delete (delete data from the back-end)
- ○ query (retrieve a list of data from the back-end)
- ○ custom (for instance "search")

Figure 43. Data operations page for WSDL.

- For resource-oriented services, such as services exposed by an SAP Netweaver Gateway, you can define multiple operations at once.



Add Service Wizard

Data Operation

Choose resource operation(s) to support

This service will be used by the application to perform certain tasks on the data displayed in the user interface. Specify the type of data operation this service provides:

□ retrieve (retrieve a single data record from the back-end)
□ query (retrieve a list of data from the back-end)
□ create (create data in the back-end)
□ delete (delete data from the back-end)
□ update (update a single data record in the back-end)

Figure 44. Data operations page for SAP

6. Click **Next** to select the **inputs and outputs** (for WSDL), or the **resource fields** (for SAP) to import.
7. Click **Finish**.

Results

The wizard closes, and one of the generated data objects and the first used operation in the list are selected in the **Data Objects** tab of the IBM Worklight Application Framework editor. You can now use this tab to customize the mapping between the generated data objects and services.

An `.xml` service description file is generated under the **services** folder of your project. This `.xml` file is used by IBM Worklight Application Framework to create data object definitions and operation mappings. An adapter is also generated in the **adapters** folder. You can use this adapter to invoke services with JavaScript calls.

The Data Object tab of the IBM Worklight Application Framework editor:

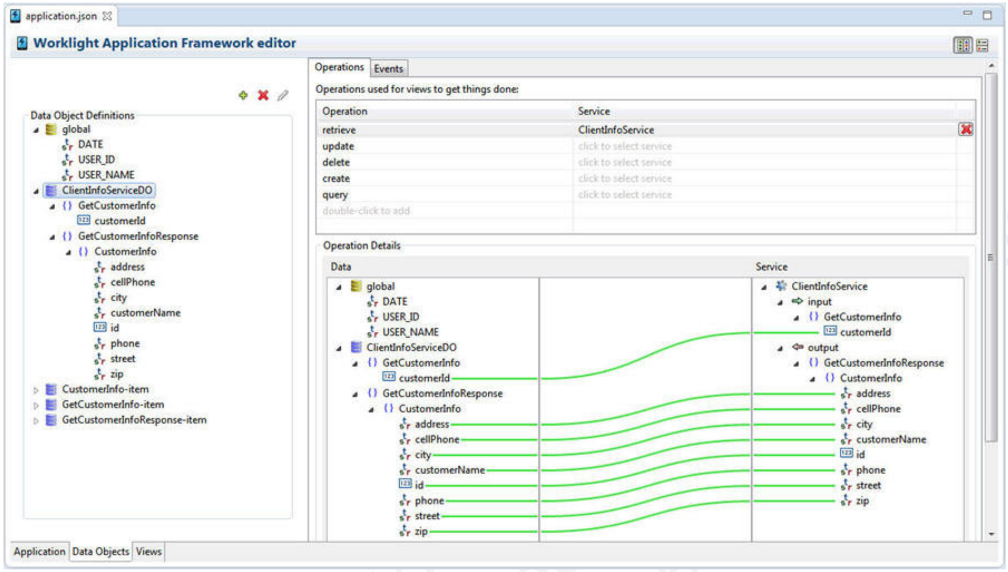
Go to the **Data Object** tab of the IBM Worklight Application Framework editor (Beta code) to create data objects, and configure data object definitions and operation mappings, as detailed in the following topics.

Before you can map data object attributes and service parameters for specific operations (such as Create, Retrieve, Update, Delete, or Query), you must first add back-end services to your project, and create data objects definitions. For more information about adding services and data objects, see “Adding services and data objects with the services discovery wizard” on page 432).

Note: You can also add data objects to your application from the **Data Object** tab (see “Adding data objects to your application from the **Data Object** tab” on page 437).

You then configure the data object definitions and you specify how to bind them to services for each selected data operation.

Figure 45. The **Data Object** *tab*



Operations used for views to get things done:

Operation	Service
retrieve	ClientInfoService
update	click to select service
delete	click to select service
create	click to select service
query	click to select service
double-click to add	

Adding data objects to your application from the **Data Object** *tab:*

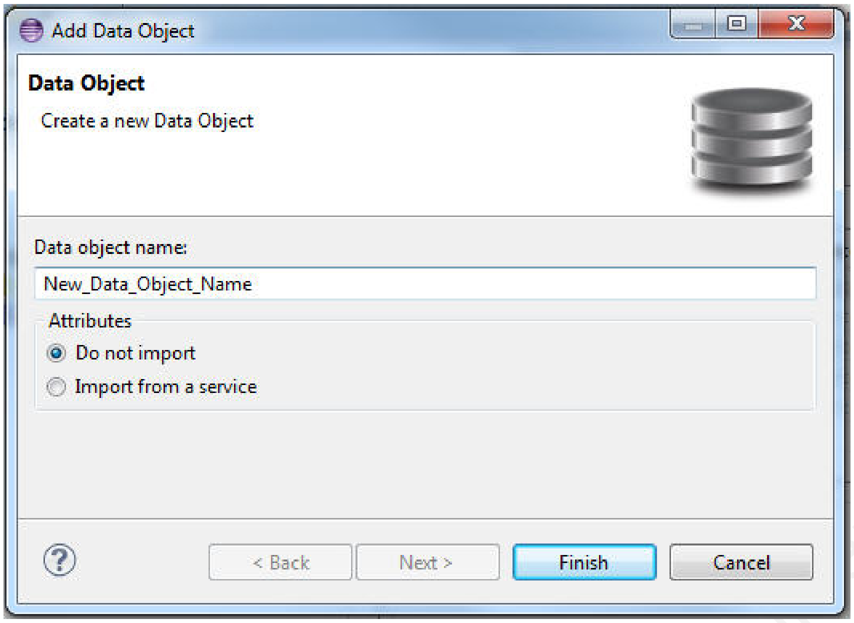
If you develop your app with IBM Worklight Application Framework (Beta code), you can create data object definitions from the **Data Object** tab of the IBM Worklight Application Framework editor.

About this task

You can define data objects for your application in two different ways: either as part of the services discovery process, as indicated in “Adding services and data objects with the services discovery wizard” on page 432, or by completing the following procedure.

Procedure

1. Double-click the `application.json` file in the `common` folder of your application to open the IBM Worklight Application Framework editor, and go to the **Data Object** tab of the editor.
2. In the **Data Object Definitions** section, select a data object entry, and click the icon ✚ . A wizard opens.
3. Enter a name for the new data object, and choose whether you want to import its attributes from a service.


Add Data Object

Data Object

Create a new Data Object

Data object name: New_Data_Object_Name

Attributes

● Do not import

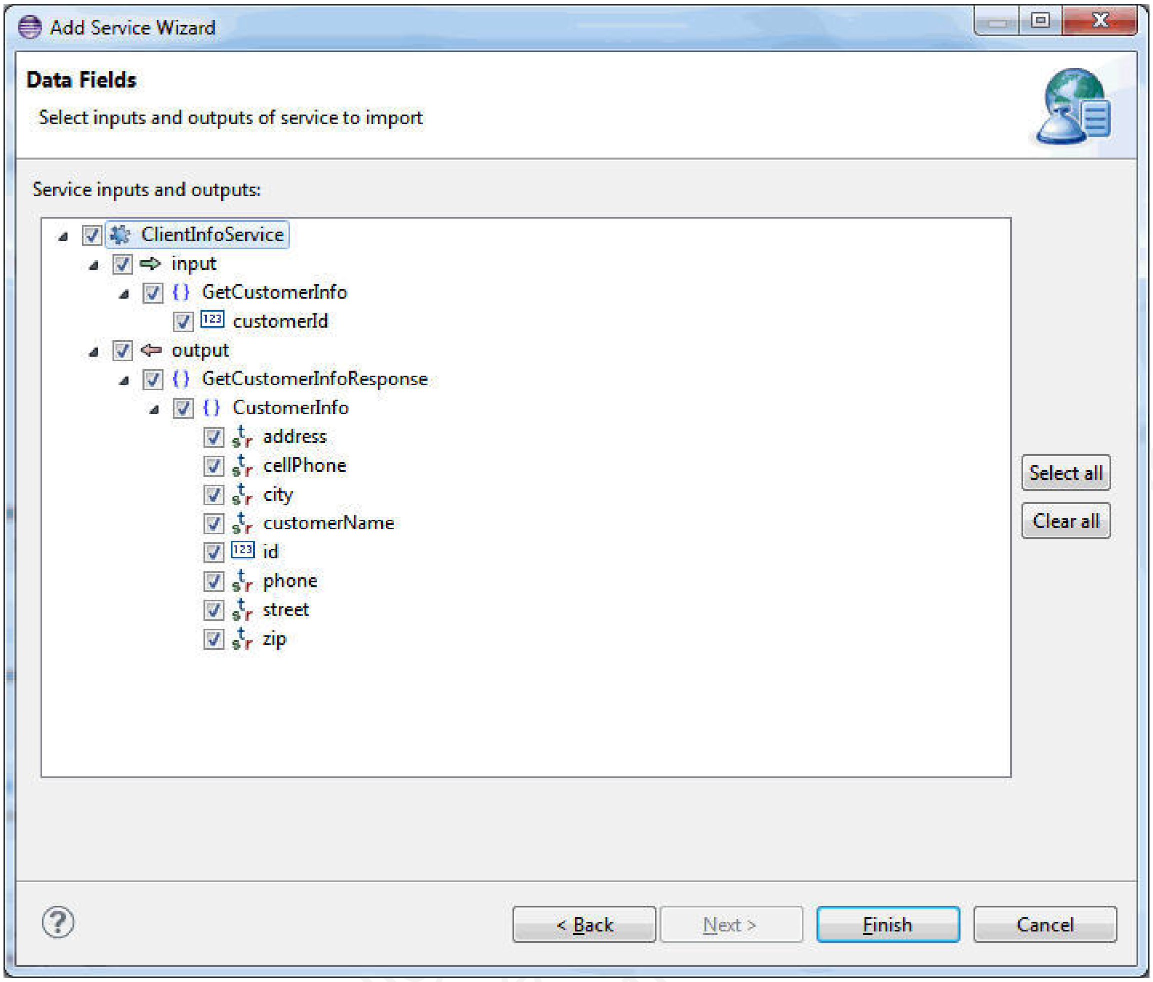
○ Import from a service

Figure 46. "Add Data Object" wizard

4. If in step 3 you selected:
   - **Do not import**, click **Finish**.

     The wizard closes, and you can see that an empty definition is added to your list of data objects. You can then add attributes manually to your new object by right-clicking its name, and clicking **New**.
   - **Import from a service**:
     a. Click **Next**.
     b. Choose a service from the list, and click **Next**.

        Note: The list is populated only if you previously added services to your project.
     c. Select the **inputs and outputs** (for WSDL), or the **resource fields** (for SAP) to import, and click **Next**.



Data Fields

Select inputs and outputs of service to import

Service inputs and outputs:

- ☑ ClientInfoService
  - ☑ input
    - ☑ GetCustomerInfo
      - ☑ customerId
  - ☑ output
    - ☑ GetCustomerInfoResponse
      - ☑ CustomerInfo
        - ☑ address
        - ☑ cellPhone
        - ☑ city
        - ☑ customerName
        - ☑ id
        - ☑ phone
        - ☑ street
        - ☑ zip

Figure 47. Example of inputs and outputs to choose for a web service

d. On the page that lists the supported data operations, select one or several types of operation available with the specified service.

Note: For WSDL-based services, you can define only one operation for each back-end service discovered.

e. Click **Finish**.

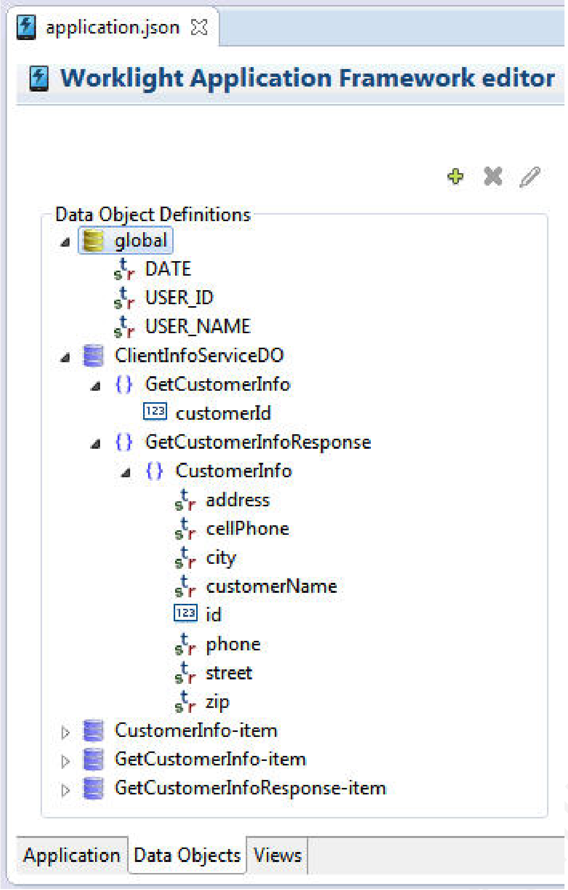
The wizard closes, and one of the generated data objects and operations are selected in the **Data Objects** tab of the IBM Worklight Application Framework editor.

Configuring data object definitions and customizing operation mapping:

After you add back-end services to your project, and define data objects and operations, you can manipulate those elements in the IBM Worklight Application Framework editor (Beta code) to customize the interactions between services and data objects.

Configuring data object definitions

The data objects that you defined are listed in the **Data Object Definitions** section of the **Data Object** tab.



application.json

Worklight Application Framework editor

```
Data Object Definitions
◢ global
      DATE
      USER_ID
      USER_NAME
◢ ClientInfoServiceDO
   ◢ {} GetCustomerInfo
         customerId
   ◢ {} GetCustomerInfoResponse
      ◢ {} CustomerInfo
            address
            cellPhone
            city
            customerName
            id
            phone
            street
            zip
▷ CustomerInfo-item
▷ GetCustomerInfo-item
▷ GetCustomerInfoResponse-item
```

Figure 48. Data Object Definitions

The **Global** entry lists available fields that you can use globally for all your operation mappings. There are three predefined global fields (**DATE**, **USER_ID**, **USER_NAME**), and more can be added during the process of mapping data objects to services, and views to data objects.

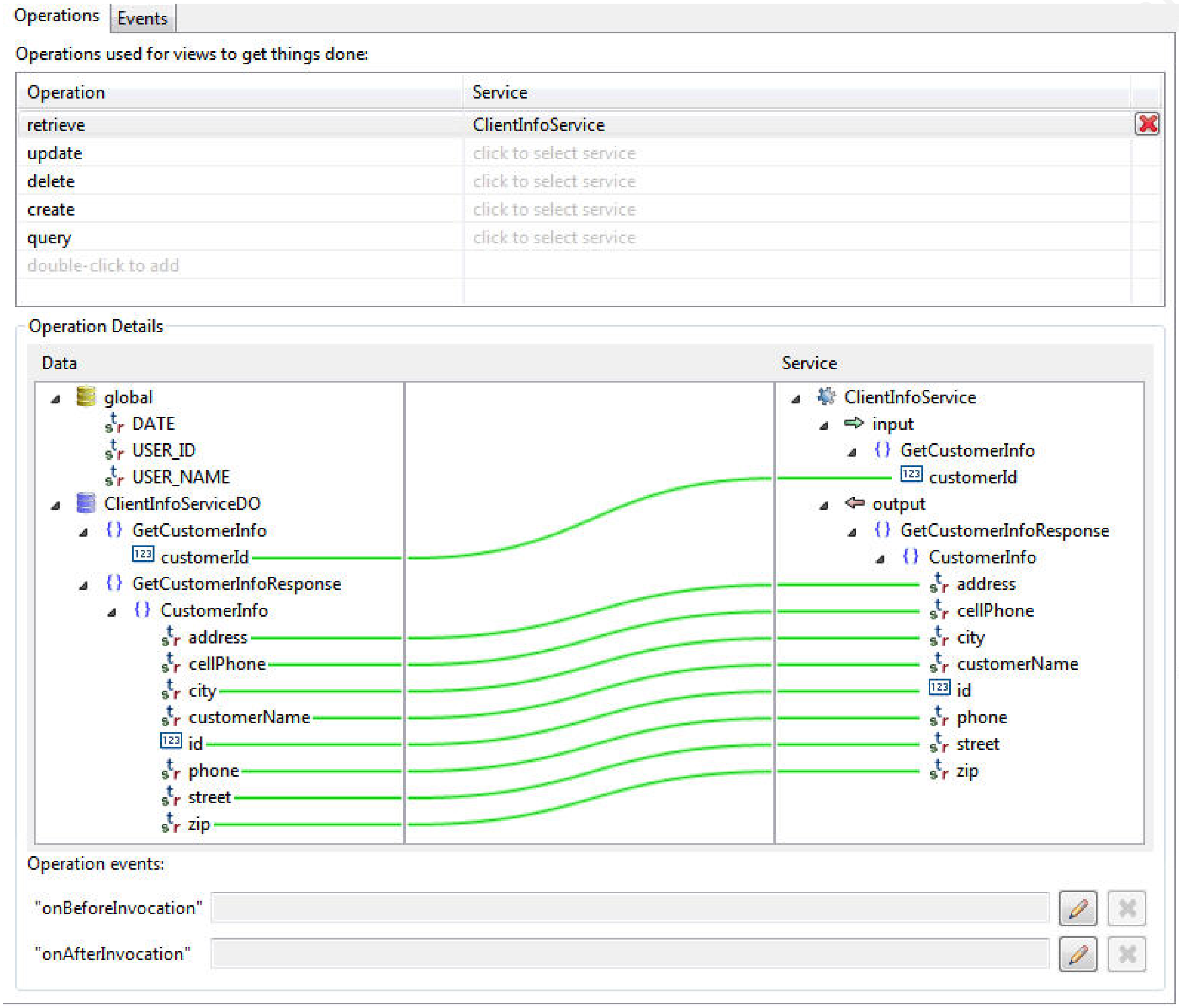
When you expand a data object entry, you can review its attributes.

- To delete a data object or an attribute, select the data object or attribute, and click the icon ✖ .
- To rename a data object or an attribute, right-click the element that you want to rename, and select **Rename**.
- To add a data object to the **Data Object Definitions** list, select any entry, and click the icon ✚ . For more information about how to create data objects from the **Data Objects** tab, see "Adding data objects to your application from the **Data Object** tab" on page 437).
- To add an attribute to a data object, right-click the data object, and click **New**.
- When you select an attribute, in the **Attribute Details** section, you can:
  - Rename an attribute, or modify its type, by entering a new name, or a new type, in the **Name** or **Type** fields.
  - Configure an event handler for the `onChange` event, by clicking the ✎ icon, and specifying a function from your JavaScript code.
- To configure event handlers for data objects:

1. Select a data object and go to the **Events** tab. Here, you can configure event handlers for the `onInit` and `onChange` events.
2. For either of these events, click the icon ✎ , and specify a function from your JavaScript code.

Customizing operation mapping

For each data object and each operation that you want to use, you can customize the mapping between the data object attributes and the service parameters in the **Operations** tab.



Operations used for views to get things done:

Operation	Service
retrieve	ClientInfoService ✖
update	click to select service
delete	click to select service
create	click to select service
query	click to select service
double-click to add	

Operation Details

Data

- global
  - DATE
  - USER_ID
  - USER_NAME
- ClientInfoServiceDO
  - GetCustomerInfo
    - customerId
  - GetCustomerInfoResponse
    - CustomerInfo
      - address
      - cellPhone
      - city
      - customerName
      - id
      - phone
      - street
      - zip

Service

- ClientInfoService
  - input
    - GetCustomerInfo
      - customerId
  - output
    - GetCustomerInfoResponse
      - CustomerInfo
        - address
        - cellPhone
        - city
        - customerName
        - id
        - phone
        - street
        - zip

Operation events:

"onBeforeInvocation" ✎ ✖

"onAfterInvocation" ✎ ✖

Figure 49. The **Operations** *tab*

1. Select a data object, and an operation from the list.

   Note: To delete operations, or add new ones, click the icon ✖ , or double-click **click to select service**, respectively, in the row of the operation you want to modify.

   - For procedural types of services, such as SOAP, only one operation is defined for each service discovered. So for this specified operation, you map a data object for the input and the output of the service.
   - For resource-based services, such as the ones exposed by SAP Netweaver Gateway, you can do the mapping for all pre-defined operations at once.

   The operation mapping is displayed in the **Operation Details** section.
2. Drag the attributes from the **Data** column to the parameters of the **Service** column, or vice versa.

   Note: To delete a link between two elements, select the green line that binds the elements, and press the Delete key.
3. Optional. In the **Operation events** section, you can configure event handlers for each operation: `onBeforeInvocation` and `onAfterInvocation`, by clicking the icon ✏ and by specifying a function from your JavaScript code.

The Views tab of the IBM Worklight Application Framework editor:

Go to the **Views** tab of the IBM Worklight Application Framework editor (Beta code) to manage the UI views of your application.

After you discover services, create data objects, and specify related operation mappings (as described in “Adding services and data objects with the services discovery wizard” on page 432 and in “Configuring data object definitions and customizing operation mapping” on page 439), you build and configure view definitions based on the data objects that you created. You can also define the flow between the screens you create.

Views are representations of the screens of your app, and of how those screens display specific data.

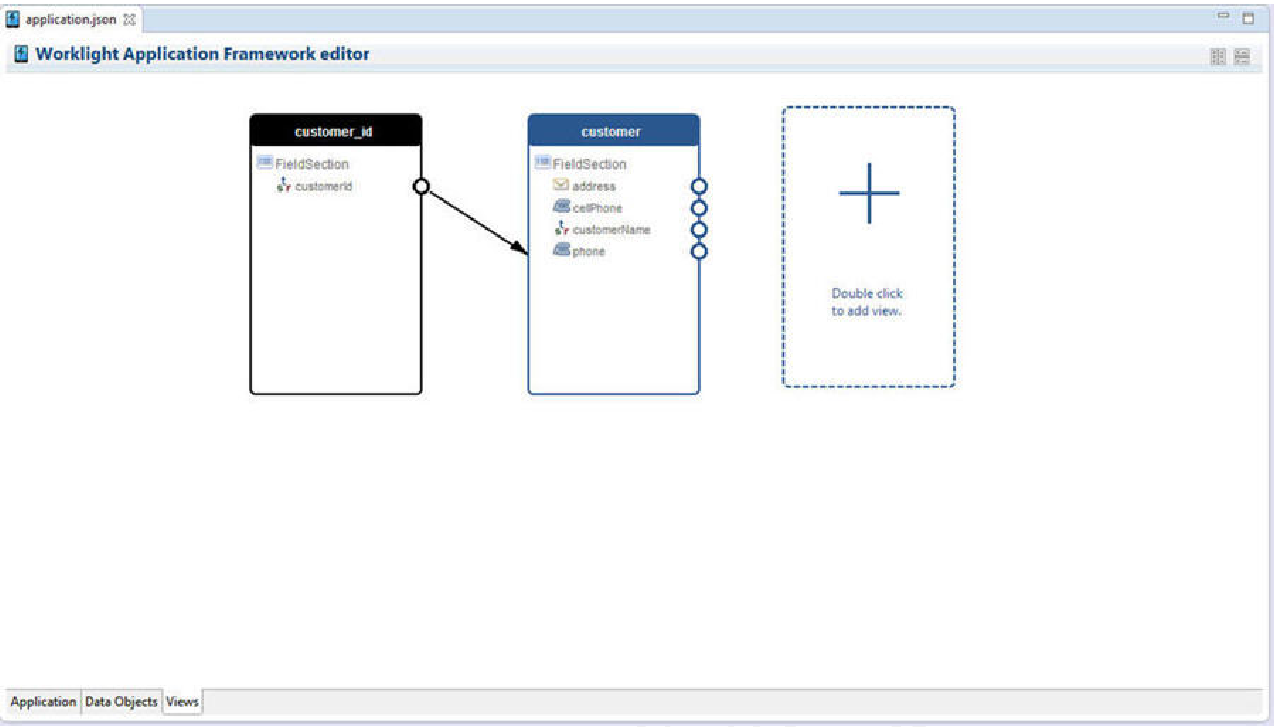


Figure 50. The **Views** *tab of the IBM Worklight Application Framework editor*

The following topics describe how to create and configure views for your application.

Adding a view to your application:

To create a view for your application, you must choose the data object, the operation to invoke on the data object, and the data object attributes to display in your view.

Procedure

1. In the **Views** tab of the IBM Worklight Application Framework editor (Beta code), double-click the rectangle with the dashed borders, or right-click anywhere in the **Views** tab and click **Add View**.

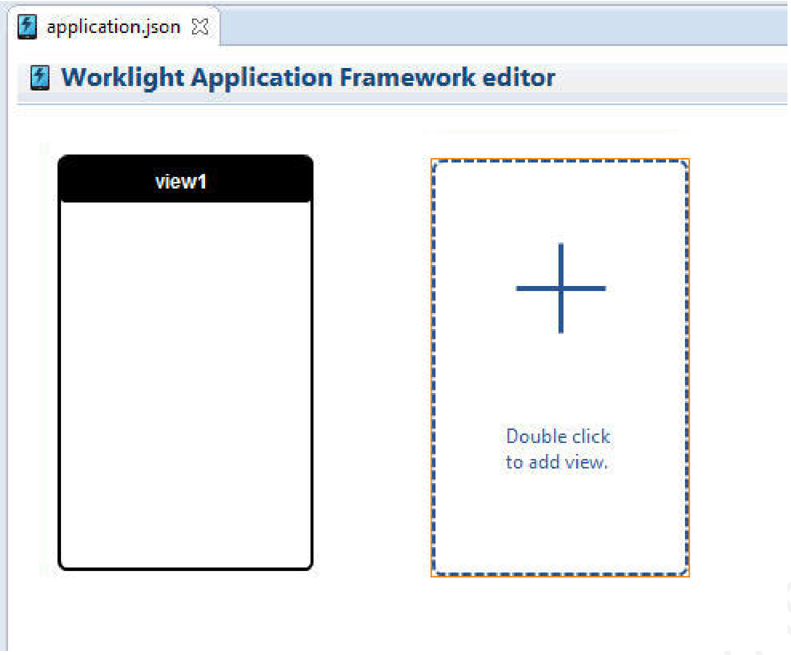
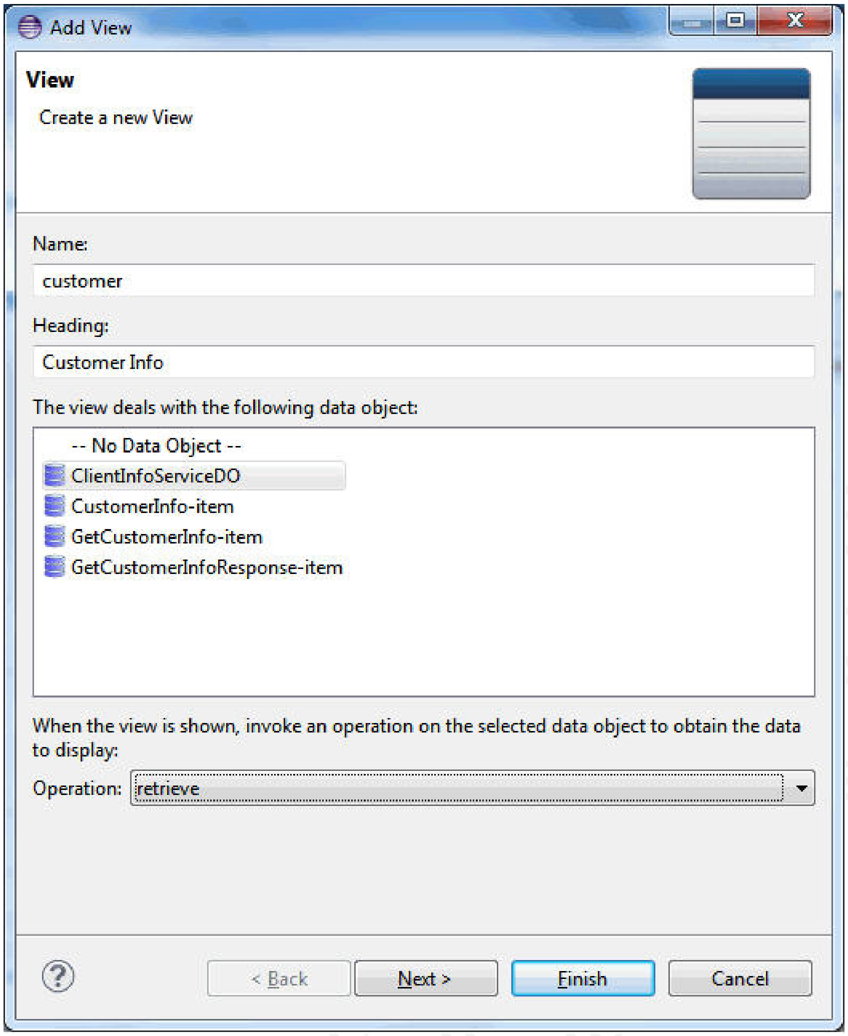


Figure 51. Create a new view

An **Add View** wizard opens.

Note: You can also access this wizard from the **Application** tab, by clicking **Create a View**.

2. Define your new view by giving it a **Name** and a **Heading**, and by specifying the data object and the operation to use.



View

Create a new View

Name: customer

Heading: Customer Info

The view deals with the following data object:

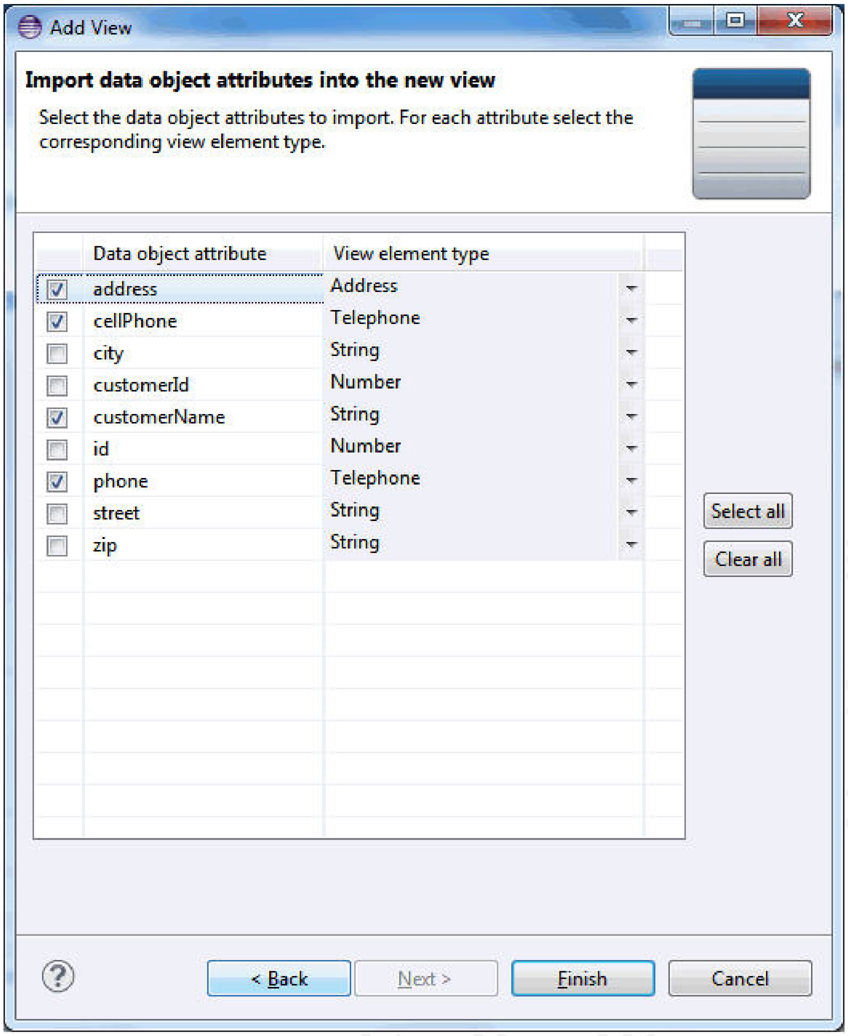
- -- No Data Object --
- ClientInfoServiceDO
- CustomerInfo-item
- GetCustomerInfo-item
- GetCustomerInfoResponse-item

When the view is shown, invoke an operation on the selected data object to obtain the data to display:

Operation: retrieve

Figure 52. Defining your new view

3. Click **Next**, and select the data object attributes to display in the new view.



Import data object attributes into the new view

Select the data object attributes to import. For each attribute select the corresponding view element type.

	Data object attribute	View element type
☑	address	Address
☑	cellPhone	Telephone
☐	city	String
☐	customerId	Number
☑	customerName	String
☐	id	Number
☑	phone	Telephone
☐	street	String
☐	zip	String

Figure 53. List of data objects attributes to import

4. Click **Finish**.

Results

The wizard closes, and you can see your new view displayed in the **Views** tab of the IBM Worklight Application Framework editor.

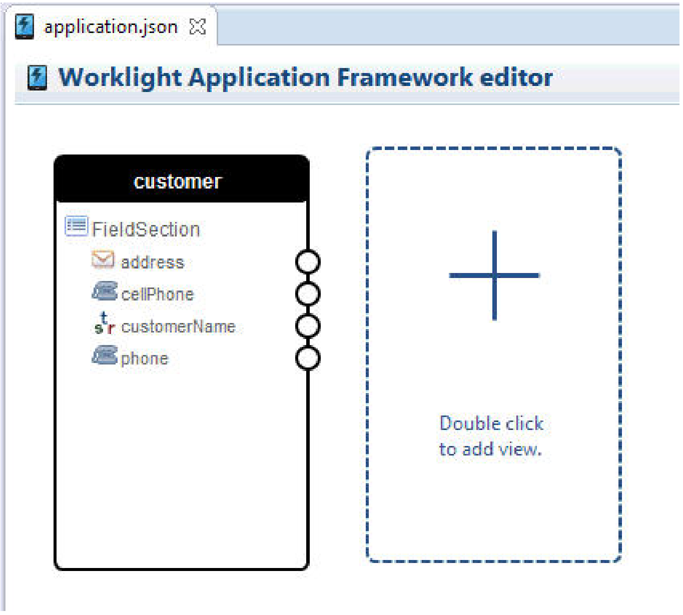


Figure 54. Result of the view creation process

Configuring the views of your application:

After you create views with the IBM Worklight Application Framework editor (Beta code), you can further configure these views and define the flow between the screens of your app.

Modifying views

When you hover your mouse over a view, a set of icons is displayed on top of the view.

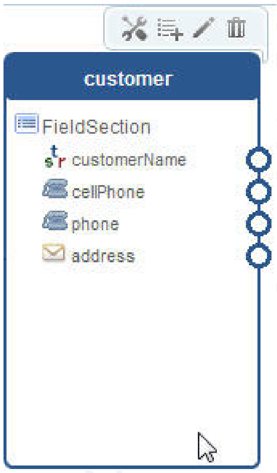


Figure 55. A view and its set of icons

To map an event handler to a view, click the icon , and specify a function from your JavaScript code.

To add a first view element to a view, click the icon , and select either **Section** or **Table** (you can add other types of view elements only inside an existing field section). You can then add other view elements in a field section by right-clicking the section and clicking **Add View Element**.

You can also further configure properties of the view element such as name, label, operations, and event handlers. To do so, you can either right-click on it and click **Configure View Element...**, or you can double-click the element.

To delete a view element and its related transition, if there is one, either select the view element and press the **Delete** key, or right-click the view element and click **Delete View Element**.

To modify the current view and configure its data objects, click the icon ✎ . In the wizard that opens, you can modify the name or the heading of the view, define operations to execute, and configure the mapping between data object attributes and views.

To delete the current view, click the icon 🗑 .

Defining an initial view

The initial view is the view that is displayed when your application starts. In the **Views** tab, the initial view is distinguishable by its black outlines. Only one view in your application can be set as the initial view.

To define a view as an initial view, right-click on the view, and click **Set as initial view**.

Creating and configuring transitions between views

After you create different views, you can create the transitions that define the navigation between those views. For example, you can specify which view comes after clicking a button. The transitions also contain information about the data objects that are transmitted from one view to another, and how this data is mapped to data objects in the target view.

To define transitions, click the circle that corresponds to the view element that you want to set as the trigger of a view transition, on the border of the view representation, and drag your mouse to the view that you want to display after the transition.

To configure an existing transition, right-click on it and click **Configure Transition**; or double-click the transition. In the new wizard that opens, you can decide whether to reuse some elements from one view to another, and map data from the source view to the target view.

To delete a transition, you can either click the transition and press the Delete key, or right-click on the transition and click **Delete Transition**.

Organizing the views layout

To tidy up the display of your existing views, right-click anywhere in the **Views** tab, and click **Organize Layout**.

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#/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#/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 wi
}
-(void)onAfterShow{
[UIView commitAnimations];
}
```

Receiving data from the web view in a Java page

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

Before you begin

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

Procedure

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

Example

Sending data from web view to the native Activity:

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

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 sc
```

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)

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.

Note: By default, IBM Worklight sets the Android application in debuggable mode in the application's manifest file. When Android runs in debuggable mode, unintended consequences can occur. One consequence is that SSL errors are not displayed by Cordova, such as when the server certificate is not trusted.

Important: When building an Android application for deployment to a production environment, do not build it to run in debuggable mode. Ensure that the `AndroidManifest.xml` file does not include an android:debuggable attribute, or set its value to `false`. For more information, see Configuring Your Application for Release.

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 56. 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 57. 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.

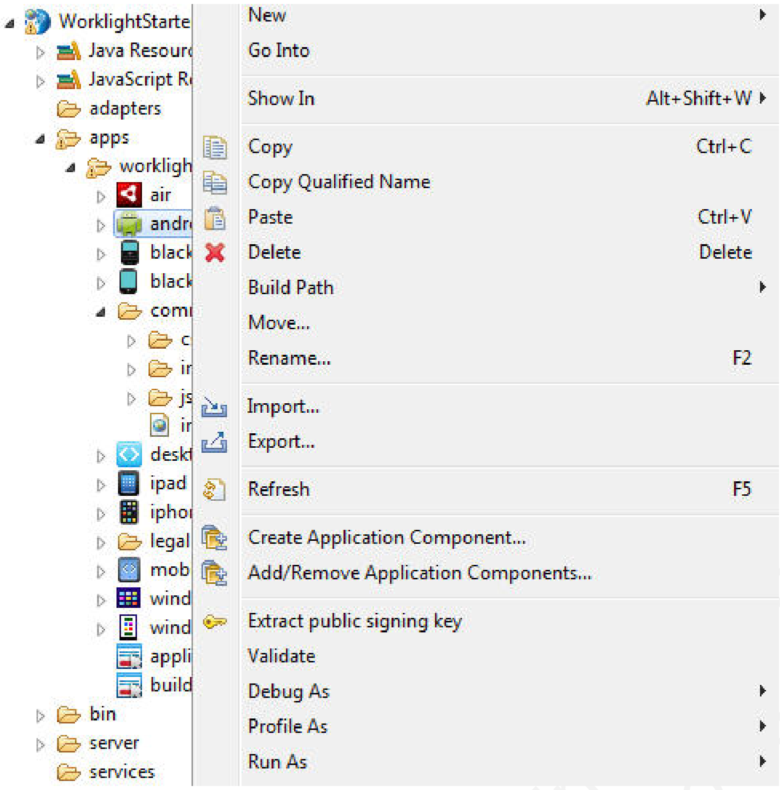
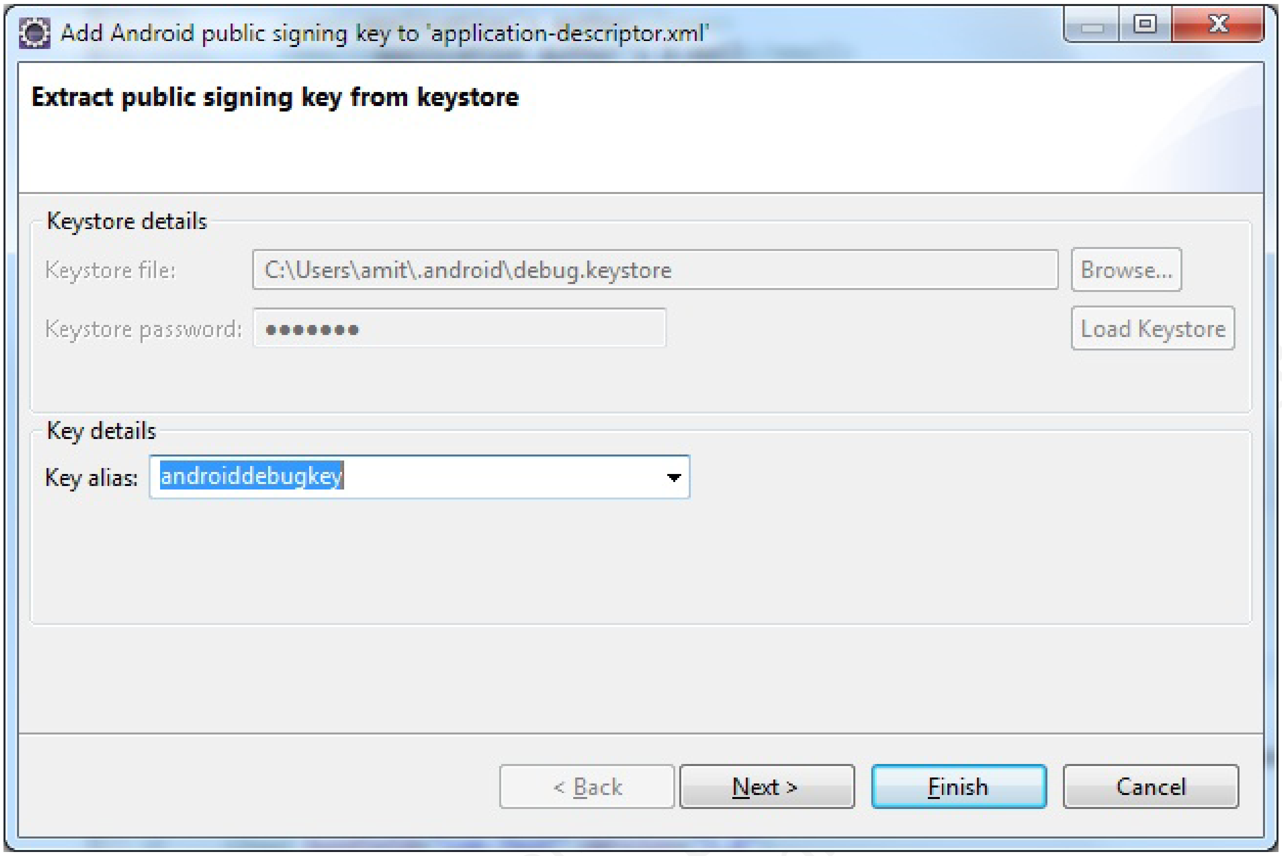


Figure 58. Extracting the public signing key

A wizard window opens.



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

Extract public signing key from keystore

Keystore details

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

Keystore password: ●●●●●●●

Key details

Key alias: androiddebugkey

Figure 59. Adding the Android public signing key

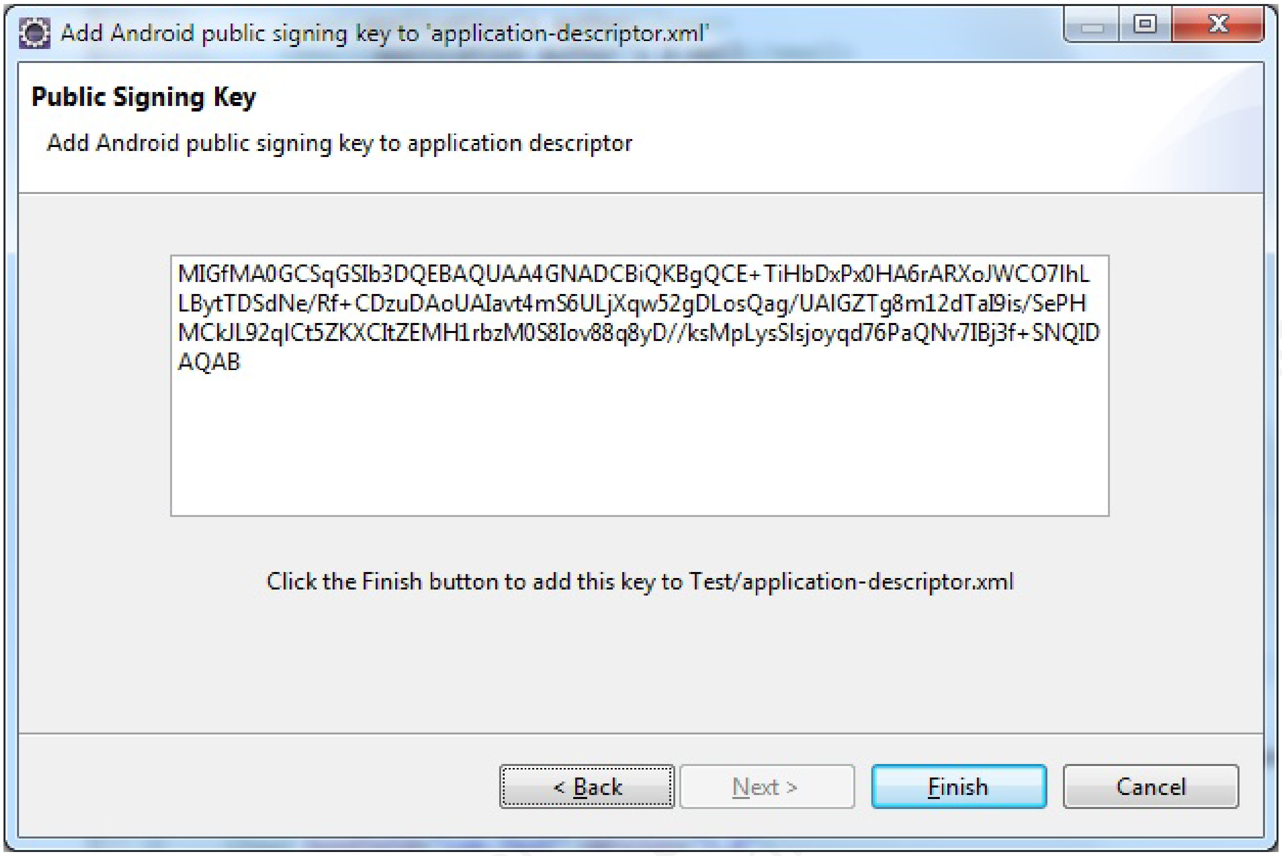
2. In this window, enter the path to your keystore. The keystore is usually in one of the following directories, according to operating system:

Option	Description

Windows XP	`C:\Documents and Settings\`*`user_name`*`\`
	`.android\`

OS X and Linux	`~/.android/`

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



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

Public Signing Key

Add Android public signing key to application descriptor

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

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

Figure 60. Android public signing key

Results

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

```
<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>MIGfMA0CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCE+TiHbDxPx0HA6rARXoJWC071hLLBytTDSdNe/
  </security>
</android>
```

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:configChang
  ```

- 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.

Building Android applications with Android Studio

From Worklight Studio, point to your Android Studio installation, and run your Android application as an Android Studio project.

About this task

You want to use Android Studio as the IDE to customize and build your Android application.

Procedure

1. In Worklight Studio, click **Window** > **Preferences** > **Worklight** (or **Eclipse** > **Preferences** > **Worklight** on Mac OS), and specify the location of your Android Studio installation.
2. Right-click the Android environment folder of your project, and click **Run As** > **Android Studio project** to start Android Studio.

Developing hybrid applications for BlackBerry

Develop hybrid applications for BlackBerry as detailed here.

IBM Worklight supports development of BlackBerry 6, 7, and 10 hybrid mobile applications.

Important: Blackberry 6 and 7 hybrid mobile application performance might not be on par with the latest BlackBerry 10 operating system due to older embedded browser technologies and hardware. It is best to use prototypes to validate that applications meet your performance targets on Blackberry 6 and 7. When advanced performance is needed, native development should be preferred.

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 3.1. 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. Install Ant Version 1.8 (or later) if it is not already installed. You can obtain Ant Version 1.8 from http://ant.apache.org/.

3. Download the `ant-contrib-1.03b.jar` file from http://central.maven.org/maven2/ant-contrib/ant-contrib/1.0b3/ant-contrib-1.0b3.jar, and save the `.jar` file in the **lib** folder of the Ant installation folder, *ANT_HOME*.
4. If you use Ant scripts, manually modify the `project.properties` file. Provide values for the following variables in `project.properties`. This step 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 debuging on the device
 qnx.device.pin=
```

5. 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.
6. 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.

   Note: Before you package or start the application with Ripple, perform the following steps:

   a. Install Ant if it is not already installed.

b. Open a command window, and navigate to the `{project name}/apps/{app name}/blackberry10/native` folder.

c. In the `{project name}/apps/{app name}/blackberry10/native` folder, run the **`ant qnx`** `copy-extensions` command.

Note: If you uninstall and install back the WebWorks SDK, make sure to run the **`ant qnx`** `copy-extensions` command again.

7. BlackBerry 10 is based on QNX. To run the application on the phone by using Cordova Ant scripts, use **`ant qnx`** `<command>`, where `<command>` is one of the commands that are defined in the `native/qnx.xml` file. For example, use **`ant qnx`** `debug-device` to build, deploy, and run the app on the device.

Worklight BlackBerry 10 project with WebWorks SDK 2.0

Follow these instructions to make a Worklight BlackBerry 10 project works with BlackBerry WebWorks SDK 2.0.

With WebWorks SDK 2.0, BlackBerry inverts their model. Instead of Cordova being the facade on top of WebWorks, WebWorks is now the facade on top of Cordova. The specific function of WebWork is implemented as Cordova plug-ins.

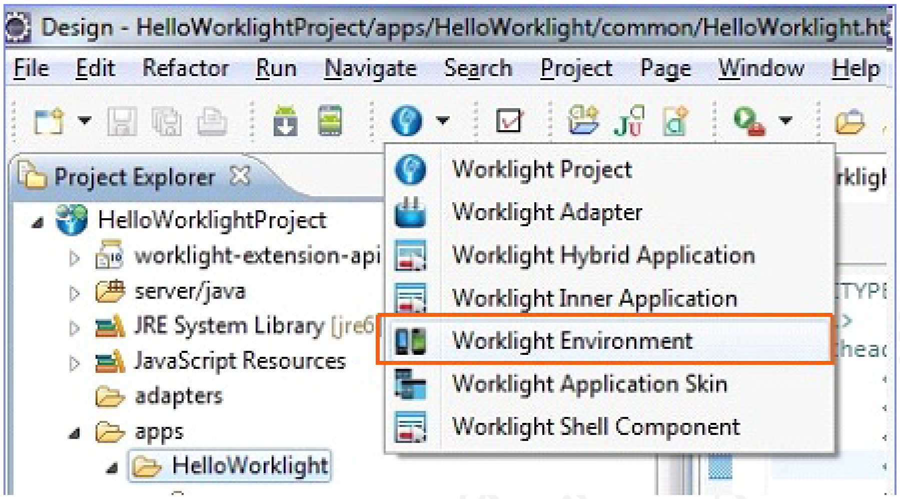
Note: WebWorks SDK 2.0 is built upon Apache Cordova 3.4 and the platform aligns with the Apache Cordova open source project now.

According to the documentation of WebWorks SDK 2.0, it basically describes how to create a new WebWorks 2.0 project and move all the assets over. For more information, see Upgrading to WebWorks 2.0.

However, there are implications on Worklight projects. The `webworks.js` file is no longer available in WebWorks 2.0. It is replaced with `cordova.js` file (BlackBerry version). There is also a difference in the project folder structure between WebWorks 1.x and WebWorks 2.0. As a result, the existing instructions in IBM Worklight do not work as-is with WebWorks 2.0. Perform the following instructions to make Worklight works with WebWorks 2.0.

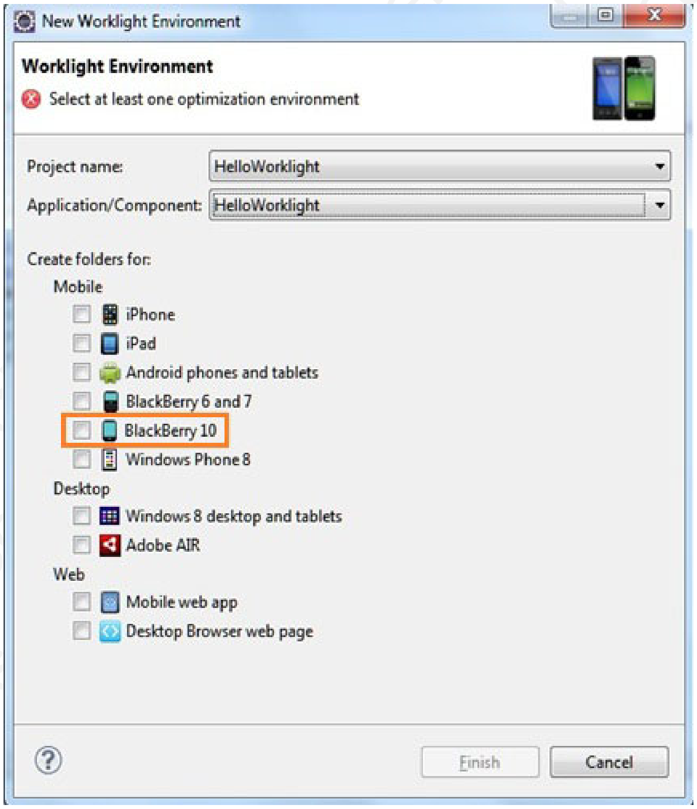
Add an environment

1. Define a **`WEBWORKS_HOME`** environment variable. The value of this variable must be the path to your WebWorks SDK.
2. Create your Worklight BlackBerry 10 application.
3. Click the **Worklight** icon, and select **Worklight Environment** to add an environment to your application.



Design - HelloWorklightProject/apps/HelloWorklight/common/HelloWorklight.ht
File Edit Refactor Run Navigate Search Project Page Window Help
Project Explorer
HelloWorklightProject
worklight-extension-api
server/java
JRE System Library [jre6
JavaScript Resources
adapters
apps
HelloWorklight
Worklight Project
Worklight Adapter
Worklight Hybrid Application
Worklight Inner Application
Worklight Environment
Worklight Application Skin
Worklight Shell Component

4. Select **BlackBerry 10**, and click **Finish**.



New Worklight Environment

Worklight Environment

Select at least one optimization environment

Project name: HelloWorklight

Application/Component: HelloWorklight

Create folders for:

Mobile

- □ iPhone
- □ iPad
- □ Android phones and tablets
- □ BlackBerry 6 and 7
- □ BlackBerry 10
- □ Windows Phone 8

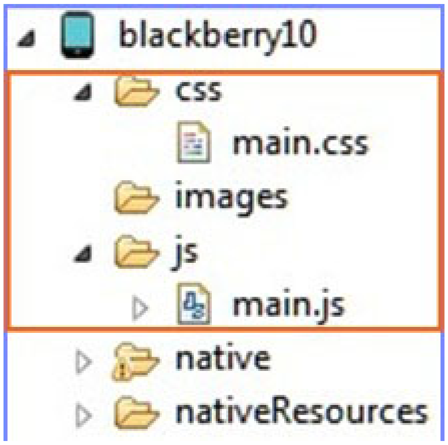
Desktop

- □ Windows 8 desktop and tablets
- □ Adobe AIR

Web

- □ Mobile web app
- □ Desktop Browser web page

5. A `blackberry10` environment folder is automatically added. This environment folder includes the following subfolders:
   - `css` – The properties that are specified here override the CSS files from the `common` folder.
   - `images` – The specific images of BlackBerry can be added here. If an image with the same file name exists in the `common` folder, it is overwritten in the BlackBerry application.
   - `js` – The JavaScript that can extend and, if required, overrides JavaScript from the `common` folder.

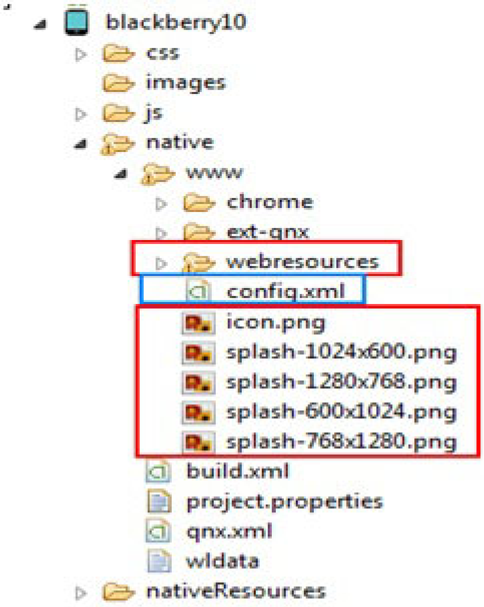


Upgrade Worklight BlackBerry 10 Project

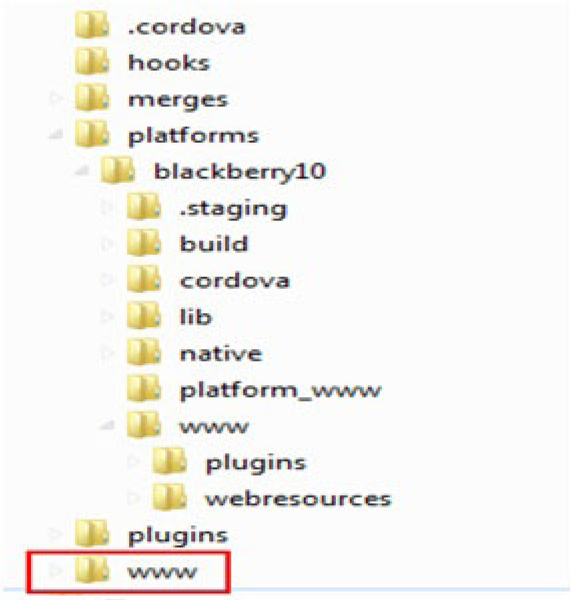
1. Install Ant. Ignore this step if Ant is installed.
2. Open a command window. Browse to the *project_name*/apps/*app_name*/blackberry10/native folder.
3. In the *project_name*/apps/*app_name*/blackberry10/native folder, run the **ant qnx upgrade-webworks-SDK-2.x** command.

Creating a project with WebWorks SDK 2.0

1. Setup BlackBerry 10 WebWorks SDK 2.0. For more information, see WebWorks: Setting up your tools.
2. Create a WebWorks SDK 2.0 project by using the following WebWorks command: `webworks create` *project_name*
3. Remove contents from *project_name*/www folder of the project.
4. Copy the webresources folder and the images under www folder of the Worklight BlackBerry 10 environment folder.



5. Paste the copied files under *project_name*/www folder of the WebWorks SDK 2.0 project.



```
.cordova
hooks
merges
platforms
    blackberry10
        .staging
        build
        cordova
        lib
        native
        platform_www
        www
            plugins
            webresources
plugins
www
```

6. Copy the `config.xml` file of the Worklight BlackBerry 10 environment folder, and paste it under the root folder of WebWorks SDK 2.0 project. Replace the existing one.

Add Device and Globalization Cordova plug-ins

Add Device and Globalization Cordova plug-ins to WebWorks project, as they are used in Worklight JavaScript files. Use the following command to add these two plug-ins:

- `webworks plugin add org.apache.cordova.device`
- `webworks plugin add org.apache.cordova.globalization`

Note: These two plug-ins must be added before you run your application, otherwise the app would not connect to Worklight Server. Add other plug-ins based on your requirement.

Basic commands to Build and Deploy

- Add plug-ins, if required.
  - `webworks plugin add` *`plugin_id`*
  -
- Build and deploy on the device.
  - Connected through USB.

    `webworks run ––devicepass` *`device_password`* `––keystorepass` *`keystore_password`*
  - Connected through wireless.

    Create target:

    - `target add` *`target_id ip-address`* `-t device -p` *`password`* `--pin` *`device_pin`*

    Then, run:

    - `webworks run --devicepass` *`device_password`* `--target=`*`target_id`* `--keystorepass` *`keystore_password`*
  - Interactive mode.

    Developer is asked to provide input for the device password, the keystore password, and so on.

    Run the command:

    - `webworks run`

- Help
  - Run the WebWorks command to get help.

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 600.

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 784. For information about how to specify IBM Worklight configuration properties, see "Configuration of IBM Worklight applications on the server" on page 772.

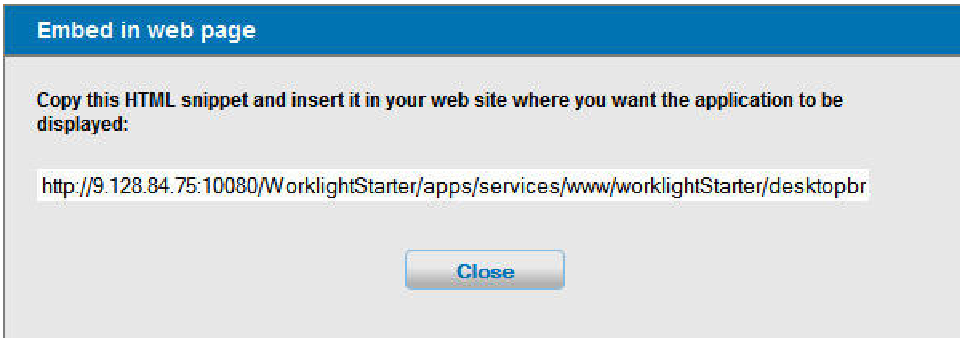
About this task

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 Worklight Console, on the Catalog tab page, 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:



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

http://9.128.84.75:10080/WorklightStarter/apps/services/www/worklightStarter/desktopbr

Figure 61. 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" on page 469.
  2. To update the client property file, see "Client property file for iOS" on page 470.
  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 471.
- For Android:
  1. To update the application descriptor file, see "Application Descriptor of Native API application for Android" on page 472.

   2. To update the client property file, see “Client property file for Android” on page 474.
   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 474.
- 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 476.
   2. To update the client property file, see “Client property file for Java Platform, Micro Edition (Java ME)” on page 477.
   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 478.

You build and deploy Native API applications by following the same procedure as for hybrid applications, by creating the `.wlapp` file and uploading it to the Worklight Console. For more information about deployment, see “Deploying applications and adapters to Worklight Server” on page 795.

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 `WLClient`.

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

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

Table 58. Properties of the `worklight.plist` file

Property	Description	Example values
`protocol`	The communication protocol with the Worklight Server.	`http` or `https`
`host`	The host name of the Worklight Server.	`localhost`
`port`	The port of the Worklight Server. If this value is left blank, the default port is used. If the **`protocol`** property value is `https`, you must leave this value blank.	`10080`
`wlServerContext`	The server URL context.	`/` **Note:** If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.
`application id`	The application ID, as defined in the `application-descriptor.xml` file.	`myApp`
`application version`	The application version, as defined in the `application-descriptor.xml` file.	`1.0`
`environment`	This property defines the IBM Worklight environment. The value of this property must be `iOSnative`. You must not modify the value of this property value.	`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 and the `worklight.plist` file of your Native API application, and copy them in a location that you can access from your native iOS project

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

2. Add the `WorklightAPI` folder and the `worklight.plist` file of your Native API application to your project.
   a. In the **Choose options for adding these files**window, select **Copy items into destination group's folder (if needed)** and **Create groups for any added folders**.

3. In the **Build Phases** tab, link the following frameworks and libraries to your project:
   - `CFNetwork.framework`
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `sqlcipher.framework`
   - `libz.dylib`
   - `libstdc++.6.dylib`

   Note: The framework `sqlcipher.framework` might already be linked.
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. On the **Build Settings** page:
   a. Click **All** (in the upper left corner) to show all settings.
   b. Add the following entry: `$(SRCROOT)/WorklightSDK/include for HEADER_SEARCH_PATH`
   c. In the **Other Linker Flags** field, enter the following value: `-ObjC`
   d. In the **Deployment** section, select a value for the **iOS Deployment Target** field that is greater than or equal to 5.0.

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 454.

`</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 59. 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. If you leave this value blank, the default port is used. If the **`wlServerProtocol`** property value is `https`, you must leave this value blank.	`10080`
`wlServerContext`	The server context.	`/` **Note:** If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.
`wlAppId`	The application id, as defined in the `application-descriptor.xml` file.	`myApp`
`wlAppVersion`	The application version, as defined in the `application-descriptor.xml` file.	`1.0`
`wlEnvironment`	This property defines the IBM Worklight environment. The value of this property must be `Androidnative`. You must not modify the value of this property value.	`Androidnative`
`GCMSenderID`	This property defines the GCM Sender ID that you must use for push notifications. By default, this property is commented.	

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, the `android-async-http.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_WIFI_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 60. 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
wlServerContext	The server context.	/ **Note:** If you use IBM Worklight Developer Edition, you must set the value of this property to the name of your Worklight project.
wlAppId	The application ID, as defined in the application-descriptor.xml file.	myApp
wlAppVersion	The application version, as defined in the application-descriptor.xml file.	1.0
wlEnvironment	This property defines the IBM Worklight environment. The value of this property must be JavaMEnative. 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.

Accelerating application development by reusing resources

Use application components, IBM Worklight project templates, and mobile patterns to accelerate the development of applications by reusing resources.

This section describes application components and Worklight project templates. For information about mobile patterns, see “Mobile patterns” on page 394.

Configuring application component and template preferences

You can configure the location of your local download folder. The download folder is the place where IBM Worklight searches for IBM Worklight project templates and application components whenever you add an application component to an IBM Worklight project or create a Worklight project from a template.

About this task

You can use the default folder *<USER_HOME>*`/IBM/templates`, or you can specify an alternative folder. If you want to use an alternative folder, you must specify it before you create application components and templates.

Procedure

1. In Worklight Studio, click **Window** > **Preferences**.
2. In the left panel, click **Worklight** > **Templates and Components**.
3. In the right panel, click **Browse**, and then select the folder that you want to use as your download folder.

Application components

Application components are reusable libraries that you can add to the applications you develop. An application component can be a client-side library or a server runtime block. Typical libraries might handle basic functions such as login or payments. They can also contain various elements such as non-visual runtime objects, visual components, integration adapters, and user interface screen packages.

Consider the example of a banking application. The application might require an image-processing library for processing checks, a non-visual runtime object, and an integration adapter to connect to the banking system for verification. A developer might consider assembling these reusable building blocks into application components, and then add them to multiple IBM Worklight projects to accelerate the development of applications for a range of different devices.

An application component can help simplify and speed up the delivery of high quality mobile applications across multiple devices. An application component can also help developers in their interactions with customers, can provide value-added services, and can help developers understand how consumers use their mobile applications.

Creating application components from IBM Worklight projects

You can create an application component based on a Worklight project. You define metadata information such as the name of the component and its version number, and you select the project resources that you want to include in the application component.

Procedure

1. From the Explorer view in Worklight Studio, right-click the Worklight project and click **Create Application Component**.
2. Provide metadata information in the fields listed in the following table:

Table 61. Application component metadata

Field	Description
Name	Name of the application component. Spaces are allowed in this field.
ID	Unique identifier for the application component. This is a read-only field. The identifier is the combination of information specified in the **Name** field (using upper case characters and without spaces) together with unique identifiers.
Author	Author or provider of the application component.
Version	Version of the application component expressed in VRM (version, release, modification) format; for example, 1.0.1.
Description	Short description of the application component.
Image	Thumbnail image that represents the application component. Supported resolution: 15x12 pixels.

Note: If you enter metadata information using certain non-Western character sets, the information might be displayed in XML encoded format in the `component.wcp` file. This does not affect the usability of the application component in any way. The characters are interpreted correctly by XML processors; for example, when you view the file using a web browser such as Firefox, it will display the correct character set.

3. From the **Application** list, select the application that you want to use as a basis for the application component, and then click **Next**.
4. In the panel that displays the project resources, select the check box next to each resource that you want to include in the application component. Consider including the files that you think would be useful as a component. Do not try to include the files that are in any case generated by default by a Worklight project.
5. Click **Browse** and specify the location and filename of the application component, and then click **Finish**. The file extension must be `.wlc` or `.zip`.

Results

The application component is created with the location and filename you specified.

What to do next

You can now view the contents of the application component and add hooks to facilitate automation.

Viewing the contents of an application component

You can open an application component to view its contents by using a file compression tool.

The application component contains folders based on the IBM Worklight project resources that were selected when the application component was created, as well as a mandatory `COMPONENT-DATA` folder.

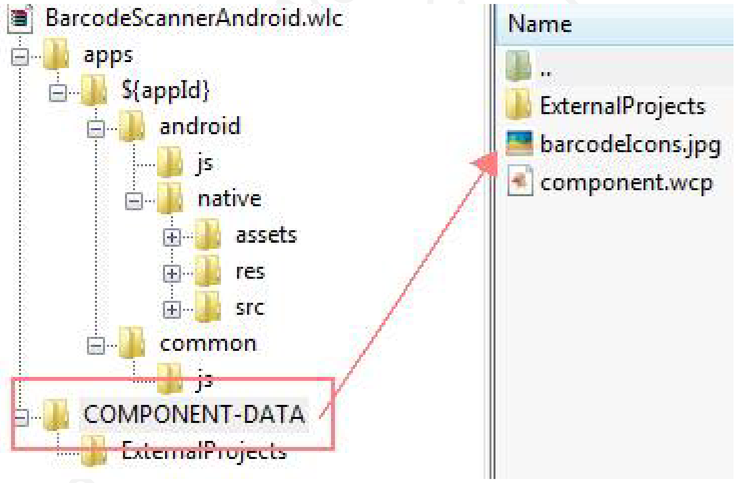


Figure 62. File structure of a typical application component

The `COMPONENT-DATA` folder contains the following files:

- The thumbnail image file that was selected when the application component was created.

- An IBM Worklight Component Processor file named component.wcp, which contains the metadata information that was specified when the application component was created.

The following contents are present in the component.wcp file:

- Component ID
- Component name
- Component author name
- Component description
- Component version
- Component thumbnail
- IBM Worklight version number

The following example shows the contents of a typical component.wcp file:

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description>
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
</ComponentData>
```

Note: Do not modify the contents of the COMPONENT-DATA folder except to add additional hooks to the component.wcp file according to the schema described in "Adding hooks to an application component."

Adding hooks to an application component

You add hooks to an application component to facilitate automation when the component is added to an IBM Worklight project. These additional hooks are optional.

To add hooks, you need to edit the component.wcp file by adding XML inner elements. If you do this directly within Worklight Studio, the edited component.wcp file is included in the next version of the application component. If you edit the component.wcp file outside Worklight Studio, you must copy the edited file manually into the Worklight Studio workspace location and then run the Create Application Component command again so that the application component is updated with the latest version of the component.wcp file.

Before you add hooks, you need to add the appropriate environment element according to the environment that the application component supports. The following table lists the element that you add for each supported environment:

Table 62. Elements for supported environments

Environment	Element
Android	<Android>
iPhone	<IPhone>
iPad	<IPad>

The following example component.wcp file includes the **Android** environment tag:

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description>
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android></Android>
</ComponentData>
```

Table 63 lists the inner elements that are supported on Android and the order in which they must appear in the schema.

Table 63. Order of inner elements for the Android environment

Order	Inner element
1	`CordovaPlugin`
2	`Activities`
3	`UserPermissions`
4	`Receivers`
5	`Strings`
6	`ExternalLibraries`
7	`Libraries`

Table 64 lists the inner elements that are supported on iOS and the order in which they must appear in the schema.

Table 64. Order of inner elements for the iPhone and iPad environments

Order	Inner element
1	`CordovaPlugin`
2	`Files`
3	`Libraries`

Note: Some hooks result in the insertion of properties in the `config.xml` file or the `AndroidManifest.xml` file when the associated application component is added to a Worklight project. Every insertion is enclosed in comments that mention the element and application component unique name. For example:

```
<!--BEGIN ANDROID CORDAVA-PLUGIN AUTOINSERTION FOR BarCodeScannerUniqueID -->

 <!--END ANDROID CORDAVA-PLUGIN AUTOINSERTION FOR BarCodeScannerUniqueID -->
```

Adding and removing Android library projects

Additional Android projects can be packaged as part of the `component.wlc` file.

Additional projects are packaged in the COMPONENT-DATA folder under the folder `ExternalProjects`. Any zip file under that folder is considered to be an "external project" and will be automatically added.

When the component is added to a Worklight project, the following things happen:

- Any additional projects are added to the Workspace.
- The additional projects are referenced from the Worklight project.
- If the external projects use a higher Android API level than is used by the Worklight Android project, the developer is prompted to upgrade to the higher API level.

When the component is removed, the following things happen:

- The additional projects are deleted from the Workspace (and from the file system).
- References to those external projects are removed from the Worklight project.

CordovaPlugin element:

This element integrates the Cordova plug-in into the application component by capturing the class name and its fully qualified name. When you add the application component to an IBM Worklight project, the CordovaPlugin properties are automatically inserted into the `config.xml` file.

Element name

`<CordovaPlugin>`

Parameters

Table 65. CordovaPlugin elements

Element	Description	Occurrences
`Name`	Name of the plug-in that uses the Cordova plug-in.	1
`ClassName`	Qualified class name of the plug-in implementation that uses the Cordova plug-in.	1

Environments supported

- Android
- iPhone
- iPad

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <CordovaPlugin>
      <Name>BarcodeScanner</Name>
      <ClassName>com.phonegap.plugins.barcodescanner.BarcodeScanner</ClassName>
    </CordovaPlugin>
  </Android>
</ComponentData>
```

Automation

When an application component that includes this element is added to a Worklight project, the `config.xml` file of the Worklight project is automatically updated to reflect the CordovaPlugin properties. The following example shows an updated `config.xml` file:

```
  <feature name="InAppBrowser">
    <param name="android-package" value="org.apache.cordova.inappbrowser.InAppBrowser"/>
  </feature>
  <feature name="Vibration">
    <param name="android-package" value="org.apache.cordova.vibration.Vibration"/>
  </feature>
  <!--BEGIN ANDROID CORDOVA-PLUGIN AUTOINSERTION FOR BarCodeScannerUniqueID -->
  <feature name="BarcodeScanner">
    <param name="android-package" value="com.phonegap.plugins.barcodescanner.BarcodeScanner"/>
  </feature>
  <!--END ANDROID CORDOVA-PLUGIN AUTOINSERTION FOR BarCodeScannerUniqueID -->
</widget>
```

Activities element:

This element enables you to add activities information to the application component. The information is declared in the Android manifest file in any IBM Worklight project that imports the component. The activities specified in the XMLContent element get appended in the Android manifest under the application element.

Element name

<Activities>

Parameters

Table 66. Activities elements

Element	Description	Occurrences
XmlContent	The XML content of the activities information that needs to be placed in the Android manifest file. You can specify one or more activity elements within the XMLContent element to be appended.	1

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <CordovaPlugin>
      <Name>BarcodeScanner</Name>
      <ClassName>com.phonegap.plugins.barcodescanner.BarcodeScanner</ClassName>
    </CordovaPlugin>
    <Activities>
      <XmlContent>
        <![CDATA[
        <!-- ZXing activities -->
        <activity
```

```
          android:name="com.google.zxing.client.android.CaptureActivity"
          android:screenOrientation="landscape"
          android:clearTaskOnLaunch="true"
          android:configChanges="orientation|keyboardHidden"
          android:theme="@android:style/Theme.NoTitleBar.Fullscreen"
          android:windowSoftInputMode="stateAlwaysHidden"
          android:exported="false">
          <intent-filter>
            <action android:name="com.phonegap.plugins.barcodescanner.SCAN"/>
            <category android:name="android.intent.category.DEFAULT"/>
          </intent-filter>
        </activity>
        <activity
          android:name="com.google.zxing.client.android.encode.EncodeActivity"
          android:label="@string/share_name">
          <intent-filter>
            <action android:name="com.phonegap.plugins.barcodescanner.ENCODE"/>
            <category android:name="android.intent.category.DEFAULT"/>
          </intent-filter>
        </activity>
        <activity
          android:name="com.google.zxing.client.android.HelpActivity"
          android:label="@string/share_name">
          <intent-filter>
            <action android:name="android.intent.action.VIEW"/>
            <category android:name="android.intent.category.DEFAULT"/>
          </intent-filter>
        </activity>
        ]]>
      </XmlContent>
    </Activities>
  </Android>
</ComponentData>
```

Automation

When a component that includes this element is added to a Worklight project, the `AndroidManifest.xml` file of the Worklight project is automatically updated to reflect the activities information. The following example shows an updated `AndroidManifest.xml` file:

```
<!--BEGIN ANDROID AUTOINSERTION FOR BarCodeScannerUniqueID -->
  <!--ZXing activities -->
  <activity android:clearTaskOnLaunch="true" android:configChanges="orientation/keyboard+hidden" .
    <intent-filter>
      <action android:name="com.phonegap.plugins.barcodescanner.SCAN"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
  <activity android:label="@string/share_name" android:name="com.google.zxing.client.android.encod
    <intent-filter>
      <action android:name="com.phonegap.plugins.barcodescanner.ENCODE"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
  <activity android:label="@string/share_name android:name="..." ...>
    <intent-filter>
      <action android:name="android.intent.action.VIEW"/>
      <category android:name="android.intent.category.DEFAULT"/>
    </intent-filter>
  </activity>
<!--END ANDROID ACTIVITY AUTOINSERTION FOR BarCodeScannerUniqueID -->
```

UserPermissions element:

This element enables you to add information about user permissions to the application component. The information controls end-user access to the native functions of a device, such as its camera or GPS functions.

Element name

`<UserPermissions>`

Parameters

Table 67. UserPermissions elements

Element	Description	Occurrences
permission	Android-specific permission constant.	1..∞

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <UserPermissions>
      <permission>android.permission.CAMERA</permission>
      <permission>android.permission.FLASHLIGHT</permission>
    </UserPermissions>
  </Android>
</ComponentData>
```

Automation

When an application component that includes this element is added to an IBM Worklight project, the `AndroidManifest.xml` file of the Worklight project is automatically updated to reflect the user permission information. The following example shows an `AndroidManifest.xml` file that is updated with user permission information:

```
  <!--BEGIN ANDROID USER-PERMISSION AUTOINSERTION FOR BarCodeScannerUniqueID -->
<uses-permission android:name="android.permission.CAMERA"/>
  <uses-permission android:name="android.permission.FLASHLIGHT"/>
  <!--END ANDROID USER-PERMISSION AUTOINSERTION FOR BarCodeScannerUniqueID -->
```

Receivers element:

This element enables you to add information about broadcast receivers to the application component. The information is declared in the Android manifest file in any IBM Worklight project that imports the component.

Element name

`<Receivers>`

Parameters

Table 68. Receivers elements

Element	Description	Occurrences
XmlContent	XML content of the broadcast receiver information that must be placed in the Android manifest file. You can specify one or more `receiver` elements within the `XmlContent` element to be appended.	1

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <Receivers>
      <XmlContent>
        <![CDATA[
        <receiver android:name="com.phonegap.plugin.localnotification.AlarmReceiver">
        </receiver>
        <receiver android:name="com.phonegap.plugin.localnotification.AlarmRestoreOnBoot" >
          <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
          </intent-filter>
        </receiver>
        ]]>
      </XmlContent>
    </Receivers>
  </Android>
</ComponentData>
```

Automation

When an application component that includes this element is added to a Worklight project, the `AndroidManifest.xml` file of the Worklight project is automatically updated to reflect the broadcast receiver information. The following example shows an `AndroidManifest.xml` file that is updated with broadcast receivers information:

```
<--!BEGIN ANDROID RECEIVER AUTOINSERTION FOR BarCodeScannerUniqueID -->
  <receiver android:name="com.phonegap.plugin.localnotification.AlarmReceiver">
  </receiver>
  <receiver android:name="com.phonegap.plugin.localnotifisation.AlarmRestoreOnBoot">
    <intent-filter>
      <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
  </receiver>
<--!END ANDROID RECEIVER AUTOINSERTION FOR BarCodeScannerUniqueID -->
```

Strings element:

This element enables you to add information about strings to the application component. The information is declared in the `android/native/res/values/strings.xml` file in any IBM Worklight project that adds the application component.

Element name

`<Strings>`

Parameters

Table 69. Strings inner elements

Element	Description	Occurrences
`XmlContent`	The XML content of the string information that needs to be placed in the `android/native/res/values/strings.xml` file. You can specify one or more `string` elements within the `XmlContent` element to be appended.	1

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <Strings>
      <XmlContent>
        <![CDATA[
        <string name="app_picker_name">Applications</string>
        <string name="bookmark_picker_name">Bookmarks</string>
        <string name="button_add_calendar">Add to calendar</string>
        ]]>
      </XmlContent>
    </Strings>
  </Android>
</ComponentData>
```

Automation

When a component that includes this element is added to a Worklight project, the `android/native/res/values/strings.xml` file of the Worklight project is automatically updated to reflect the strings information. The following example shows an updated `strings.xml` file:

```
<!--BEGIN ANDROID STRING AUTOINSERTION FOR BarCodeScannerUniqueID -->
  <string name="app_picker_name">Applications</string>
  <string name="bookmark_picker_name>Bookmarks</string>
  <string name="button_add_calendar">Add to calendar</string>
<!--END ANDROID STRING AUTOINSERTION FOR BarCodeScannerUniqueID -->
```

Libraries element (Android):

This element enables you to add required libraries to the application component. The libraries are added to the `android\native\libs` folder in any IBM Worklight project that imports the component.

Element name

`<Libraries>`

Parameters

Table 70. Libraries elements

Element	Description	Occurrences
`Path`	Path to the archive file relative to the root location of the application component. The libraries should be kept only in the `COMPONENT_DATA` folder.	1..∞

Note: Do not copy libraries to the `ExternalProject` folder.

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <Libraries>
      <Path>zxing.jar</Path>
    </Libraries>
  </Android>
</ComponentData>
```

Automation

When an application component that includes this element is added to a Worklight project, the `android\native\libs` folder of the Worklight project is automatically updated to reflect the dependent libraries information. Figure 63 shows an updated android\native\libs folder based on the example <Libraries> element.

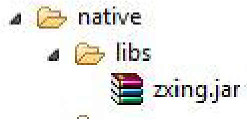


Figure 63. Example of updated libs folder

ExternalLibraries element:

This element enables you to add information about external libraries to the application component. The information provides pointers to external libraries that are to be downloaded by the developer who adds the application component to an IBM Worklight project. Typically, these libraries are additional libraries that are not packaged as part of the application component.

Element name

<ExternalLibraries>

Parameters

Table 71. ExternalLibraries elements

Element	Description	Occurrences
URL	URL from where the external library can be retrieved.	1
Message	Instructional message about addition of external libraries that need to be displayed to the developer after importing the component.	1

Environments supported

- Android

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <Android>
    <ExternalLibraries>
      <URL>http://get-library-here.com</URL >
      <Message>Please download VeryUsefulLibrary and copy into the native folder of your project</Mes
    </ExternalLibraries>
  </Android>
</ComponentData>
```

Automation

When an application component that includes this element is added to a Worklight project, a dialog box displays the specified message.

Files element:

This element enables you to add information about files that are required for the application component to work. These files are required by the native project of iPhone or iPad. The information specifies files that need to be added to the Xcode project in an IBM Worklight project. The information is displayed as a list in a dialog box whenever a developer adds the application component to a Worklight project. The files are not actually copied; instead, the developer is prompted to copy them.

Element name

`<Files>`

Parameters

Table 72. Files elements

Element	Description	Occurrences
`file`	Name of each file.	1..∞

Note: The dialog box only displays the list of files to be added to the XCode project. These files must be added manually to the XCode project. The files must be included as part of the application component.

Environments supported

- iPhone
- iPad

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <IPhone>
    <Files>
      <file>CDVBarcodeScanner.mm</file>
      <file>zxing-all-in-one.cpp</file>
    </Files>
  </IPhone>
</ComponentData>
```

Automation

When a component that includes this element is added to a Worklight project, a dialog box displays a message prompting the developer to add the listed files to the Xcode project manually.

Note: Automatic modification of Xcode projects after adding components is currently not supported.

Libraries element (iPhone and iPad):

This element enables you to add information about libraries to be added to the application component. The information specifies libraries that need to be added to the Xcode project in an IBM Worklight project. The information is displayed as a list in a dialog box whenever a developer imports the application component into a Worklight project.

Element name

`<Libraries>`

Parameters

Table 73. Libraries elements

Element	Description	Occurrences
`library`	Name of each library.	1..∞

Note: The dialog box only displays the list of libraries to be added to the XCode project. These libraries must be added manually to the XCode project.

Environments supported

- iPhone
- iPad

Example

```
<ComponentData>
  <ID>BarCodeScannerUniqueID</ID>
  <Name>Barcode Scanner Android</Name>
  <Author>IBM</Author>
  <Description>Barcode Scanner for Android by IBM</Description >
  <Version>1.0.0</Version>
  <Image>barcodeIcons.jpg</Image>
  <WLVersion>6.1.0</WLVersion>
  <IPhone>
    <Libraries>
      <library>CoreVideo.framework</library>
      <library>AVFoundation.framework</library>
    </Libraries>
  </IPhone>
</ComponentData>
```

Automation

When a component that includes this element is added to a Worklight project, a dialog box displays a message prompting the developer to add the listed libraries to the Xcode project manually.

Note: Automatic modification of Xcode projects after adding components is currently not supported.

Validating application components

After creating an application component and adding hooks, you must validate the `component.wcp` file to ensure that it conforms to the correct syntax.

About this task

Follow this procedure to validate the component.wcp file.

Procedure

1. Import the application component into an IBM Worklight project. The component.wcp file is validated during import.
2. Look for error messages in the Worklight Console. The following example error message indicates that the ClassName parameter is not specified in the CordovaPlugin element:

   ```
   Component = BarcodeScannerIOS.wlc
   Found component.wcp
   In component.wcp, one or more property values(s) are empty/invalid. For details, see below:
   Reason:cvc-complex-type.2.4.b: the content of element 'CordovaPlugin' is not complete. One of
   ```
3. Correct the errors by using information in the message text and by referring to the topics under "Adding hooks to an application component" on page 481 that describe the syntax.

Adding application components to IBM Worklight projects

After you have created and validated application components, you can add them to your Worklight projects.

Before you begin

Before you add application components, you need to complete the following tasks:

1. Create a Worklight project with an initial hybrid application. You can only add application components to Worklight projects that have been created with an initial hybrid application (See "Creating IBM Worklight projects" on page 316.)
2. Set up new IBM Worklight environments that the application components require. If you do not add all the required environments, error messages are displayed when you add application components. (See "Setting up a new IBM Worklight environment for your application" on page 338.)

   Note: Adding an environment takes some time before it is deployed. Ensure that the environments you add are deployed before you proceed to the next step.
3. Copy the application component files to your download folder. (For information about specifying a download folder, see "Configuring application component and template preferences" on page 478.)

About this task

Automatic modification of Xcode projects after adding components is currently not supported.

Procedure

1. In the Explorer view in Worklight Studio, right-click the application and click **Add/Remove Application Component(s)**. The Add/Remove Application Component(s) window opens.
2. From the **Application** list, select the relevant application.
3. Select the check box for each application component you want to add, and then click **Finish**.

Results

The selected application components are added to the Worklight project. Each application component file is marked with the application component identifier.

Removing application components from IBM Worklight projects

You can remove application components from a Worklight project if they are no longer required.

About this task

Automatic modification of Xcode projects after removing components is currently not supported.

Procedure

1. In the Explorer view in Worklight Studio, right-click the application and click **Add/Remove Application Component(s)**. The Add/Remove Application Component(s) window opens.
2. From the **Application** list, select the relevant application.
3. Clear **In Use** for each application component you want to remove, and then click **Finish**. In some cases, an information message might be displayed, or you might be prompted to confirm that you want to remove an application component.

Results

The application component files are removed from the Worklight project, and the project is restored to its former state.

Troubleshooting adding and removing application components

Whenever you add or remove an application component, the existing Worklight project files are backed up.

The backup of each original file is named `<orig filename>.backup_<add/remove>_<component id>_<date and time>`, where:

- `<orig filename>` is the full name of the file being modified (for example, `AndroidManifest.xml`).
- *<add/remove>* is the word “add” or the word “remove” according to the operation being performed.
- *<component id>* is the ID of the component being added or removed.
- *<date and time>* is the timestamp of the operation in the format `YYYYMMDD_HHMMSS`.

For example, when adding the barcode scanner for the Android component, the file `config.xml` is backed up to `config.xml.backup_add_BarCodeScannerUniqueID_20131015_190032`.

IBM Worklight project templates

Worklight project templates enable you to accelerate the development of applications by not having to start from scratch. You can use Worklight project templates to provide value added services and you can add elements that are consistent with the look and feel of your brand.

Creating IBM Worklight project templates

You can create Worklight project templates by exporting IBM Worklight projects. You define metadata information such as the name of the template, and you select the Worklight project that you want to use as the basis for the template.

Before you begin

If you want to identify the source code that can be configured by developers who use the Worklight project template, add FIX Me task tags in the configurable source code before you create a template.

About this task

The following limitations apply:

- Only hybrid Worklight projects can be used as a basis for Worklight project templates.
- You can create Worklight project templates only from Worklight projects that contain single applications.

Procedure

1. From the Explorer view In Worklight Studio, right-click the required Worklight project, and then click **Export**.
2. Expand **IBM Worklight**, select **Worklight Project Template**, and then click **Next**.
3. Provide information in the fields listed in the following table, and then click **Finish**:

Table 74. Worklight project template metadata

Field	Description
Template Name	Name of the Worklight project template. Spaces are allowed.
Author	Author or provider of the Worklight project template.
Description	Brief description of the Worklight project template.
Thumbnail	Thumbnail image to identify the Worklight project template. Valid file formats: `.jpg`, `.jpeg`, `.png`, `.gif`. Maximum size: 40x40 pixels.
Template Archive	Location and filename of the template. Valid filename extensions: `.wlt` and `.zip`.

Results

The Worklight project template is created with the location and filename you specified.

Viewing IBM Worklight project templates

You can open a Worklight project template to view its contents by using a file compression tool.

The Worklight project template contains folders taken from the Worklight project on which it is based, as well as a mandatory `TEMPLATE-DATA` folder.

Note: Do not modify the contents of the TEMPLATE-DATA folder.

The TEMPLATE-DATA folder contains the following files:

- The thumbnail image file that was selected when the Worklight project template was created.
- A template properties file named template.properties, which contains the metadata information that was specified when the Worklight project template was created.

The following contents are present in the template.properties file:

- Template title
- IBM Worklight version used to create the template
- Template ID
- Template author
- Template description
- Template thumbnail
- IBM Worklight version number

The following example shows the contents of a typical template.properties file:

```
title-Simple RSS Reader
wl_version=6.1.0.
id=RSS-09e1ac62-5c34-432e-8597-c6349eade74c
author=IBM
description=Displays entries from an RSS feed (www.example.com). The user can click an entry to read
image=rss_reader3.png
```

Creating IBM Worklight projects from IBM Worklight project templates

You can use Worklight project templates to create Worklight projects. You can select templates that are available in your download folder.

Before you begin

If the owner of a Worklight project template has provided FIXME tasks, they are displayed in the Tasks tab. To view them, you need to enable the FIXME tasks view:

1. In Worklight Studio, click **Project** > **Properties** > **General** > **Editors** > **Structured Text Editors** > **Task Tags**.
2. Click **Enable searching for Task Tags**.
3. Click **Apply** > **OK**. The Task tab is displayed. When you create an IBM Worklight project from an IBM Worklight project template, the Task tab lists any FIX Me tasks to be completed.

Procedure

1. In Worklight Studio, click **File** > **New** > **Worklight Project**. The New Worklight Project window opens.
2. In the **Name** field, enter a name for the Worklight project.
3. From the **Project Templates** pane, click **Shared Templates**, and then click **Next**. For information about the Hybrid Application, Inner Application, Native API, and Shell Component options, see “Creating IBM Worklight projects” on page 316.
4. In the **Application name** field, enter a name for your application, and then click one of the templates available from the list. The list displays templates

available in the download folder. (For information about configuring the download folder, see “Configuring application component and template preferences” on page 478.)

5. To complete the creation of the Worklight project from the highlighted Worklight project template, click **Finish**. The Design perspective is opened. The Tasks view shows the FIX Me tasks available (if any) within the template content. Links are provided to positions within the template, and accompanying descriptions explain what is configurable.
6. Optional: Review the list of FIX Me tasks available in the Tasks view, and apply any appropriate fixes.

Results

The new Worklight project is added to the Explorer view in Worklight Studio. The project includes all resources that are included in the selected template.

Building and deploying in Worklight Studio

After you create your IBM Worklight project, you must build the application and deploy it to Worklight Server to run and test it.

Previous sections (“Developing hybrid and web applications” on page 322 and “Developing native applications” on page 467) covered the basics of creating and working with your projects in Worklight Studio. This topic provides an overview of the process that is used to build these projects, deploy them to an instance of Worklight Server, and run them.

During development, you usually perform this action by right-clicking your application or an environment in the Project Explorer view of Worklight Studio, selecting **Run As** from the menu, and selecting one of its options.

Each of these menu options is covered in the topics that follow the overview of the build process below.

The IBM Worklight build process

Building a Worklight application for a specific environment (for example, iOS or Android) is the process that transforms the JavaScript, HTML and CSS code that you have created for your application into a mobile application for the specified target. The build process produces several elements:

- A native project for the target platform that is stored in the `native` folder of the environment.
- A Worklight application file (`.wlapp`) that contains the application metadata and Web resources that are used by Worklight Server to identify and service the mobile application.

The native project that is generated depends on the target environment:

- For the iPhone and iPad environments, the build operation creates a native Xcode (the native IDE for building iOS applications) project, which is placed under the `native` folder of the environment. You can then use Xcode to build the final iOS application. If you are working in Worklight Studio on a Mac, you can also right-click the `iphone` or `ipad` environment and use the **Run As** > **Xcode project** option to build the environment and open Xcode for that project. For more information about the iOS development environment, see the module *Previewing your application on iOS*, under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 27.

- For the Android environment, the `native` folder under the `android` folder contains automatically-generated Android application code that is imported into the Eclipse workspace as an Android project. You use the ADT plugin (Android Development Tools for Eclipse) to build the final Android application. If you are working in Worklight Studio on a computer with ADT installed, you can also right-click the `android` environment and use the **Run As** > **Android Studio project** option to open that project. For more information about the Android development environment, see the module *Previewing your application on Android*, under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 27.
- For BlackBerry applications, the `native` folder contains BlackBerry code that you can compile using the Ripple development environment. For more information about the BlackBerry development environment, see the modules *Previewing your application on BlackBerry 6 and 7* and *Previewing your application on BlackBerry 10*, under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 27.
- For Windows Phone, the `native` folder contains a Visual Studio project that you compile to build the final Windows Phone application. For more information about the Windows Phone development environment, see the module *Previewing your application on Windows Phone 8*, under category 2, *Hello Worklight*, in Chapter 3, “Tutorials and samples,” on page 27.

The resulting files from the build are stored in the Worklight Studio project hierarchy in the `<project_name>\bin` directory, using the following naming conventions:

- `<project_name>.war` – the WAR file for the project.
- `<project_name>-app.all.wlapp` – a WLAPP file containing all environments for the project.
- `<project_name>-app-common.wlapp` – a WLAPP file containing all common resources for the project.
- `<project_name>-app.<environment_name>.wlapp` – a WLAPP file for each environment in the project; for example: `MyProject-app-android.wlapp`.

Note: Only the latest build is contained in the `<project_name>\bin` directory at any time. If you create multiple builds for different target servers for deployment with the Worklight Console, with the supplied Ant tasks, or using the Worklight Server Configuration Tool, you must deploy them after each build operation, because the next build will overwrite the existing files.

You can modify the generated native projects if, for example, you want to add native code or Cordova plugins to the application. If you modify the HTML, JavaScript, CSS, the application descriptor file, or any application resources, you must rebuild the environment using the appropriate **Run As** > **Build ...** command to update the Worklight application file (`.wlapp`) and the native project.

Once the build of the application completes, it is deployed or not deployed depending on the **Run As** command used:

- The **Run As** > **Run on Worklight Development Server** option deploys the application metadata and Web resources (the `.wlapp` file) to the internal Worklight Development Server. If you have defined an alternative test server using the **Run As** > **Build Settings and Deploy Target** option, you can also deploy directly to that instance of Worklight Server. In this case, the command name changes to include the name of the target server; for example, **Run As** > **Run on Tomcat Test Server**.

- The **Run As** > **Build ...** options (the exact command name may change depending on context) builds the application or environment but does not deploy it. Deployment is done through the Worklight Console, through supplied Ant scripts, or using the Worklight Server Configuration Tool. For more information on deploying Worklight apps to remote servers, see Chapter 10, “Deploying IBM Worklight projects,” on page 711.

The following topics go into more detail on each of the **Run As** menu options.

The Run on Worklight Development Server command

Information about the menu command that is used to build and run an application or environment on your designated test server.

About this task

You use this menu option when you want to build your project in Worklight Studio and run it on the internal Worklight Development Server. This instance of Worklight Server is created automatically when you install Worklight Studio, and is described in “The Worklight Development Server and the Worklight Console” on page 343.

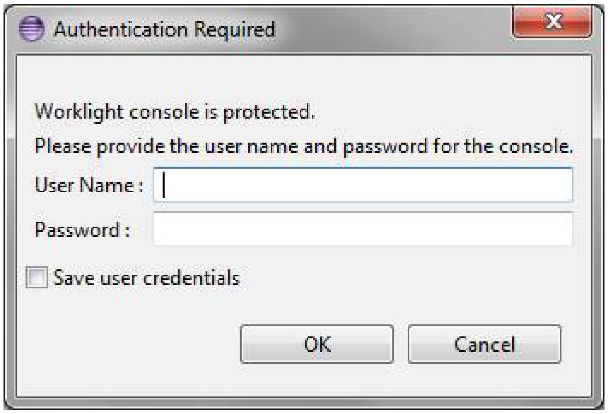
Note: This command name is context-sensitive in that it does not always display the name **Run on Worklight Development Server**. If you use the Configure Worklight Build and Deploy Target dialog to choose a different test server, the name of the server that is selected in the **Worklight server to test applications** area is displayed in the command name instead. For example, if you add a local instance of Worklight Server running on Apache Tomcat, name it "Tomcat Test Server" and designate it as your default test server, this command appears as **Run As** > **Run on Tomcat Test Server**. For consistency, when referring to this menu command in this IBM Worklight user documentation, the default name of this command (**Run on Worklight Development Server**) is used throughout.

Procedure

When you choose the **Run As** > **Run on Worklight Development Server** option, Worklight Studio performs the following actions:

- It starts the Worklight Development Server, if it is not already running.
- It builds your app and all of its included environments.
- It deploys the app to the Worklight Development Server (or designated test server), reporting success, failure, and any error messages in the Console view of Worklight Studio.
- When you open the Worklight Console that is associated with the Worklight Development Server (or designated test server), that console displays the successfully deployed app and all of its environments. The console also displays detailed build and deployment messages, if required.

Note: If your Worklight Console is secured, an Authentication Required dialog appears, prompting you to enter the **User Name** and **Password**:


Authentication Required

Worklight console is protected.

Please provide the user name and password for the console.

User Name :

Password :

□ Save user credentials

If you enter the correct credentials, the deployment continues when you click **OK**.

The dialog also contains a **Save user credentials** check box. If you select it before you click **OK**, the credentials are stored in Eclipse secure storage. This information can be edited in Eclipse by selecting **Preferences** > **General** > **Security** > **Secure Storage**. If you enter incorrect credentials, the deployment fails and the credentials are not saved.

You can also use this menu command to run and test individual Worklight environments by right-clicking on the environment in the Project Explorer and clicking **Run As** > **Run on Worklight Development Server**. This option runs and tests the selected environment on the designated test server.

Important: This menu command always uses as its target server the Worklight Server instance currently selected in the **Worklight server to test applications** area of the Configure Worklight Build and Deploy Target dialog. It ignores any server information that is entered in the **Build the application to work with a different Worklight server** area of the Configure Worklight Build and Deploy Target dialog.

Troubleshooting Worklight Development Server startup

If the Worklight Development Server fails to start and sends a timeout error message, consider increasing the default timeout value or remove all applications that you do not intend to work on.

Symptom

When you try to start or run an application on a Worklight Development Server, it sometimes fails to start and sends the following message:

```
Server Worklight Development Server was unable to start within 60 seconds. If the server requires mo
```

Cause

IBM Worklight Studio sets a 60-second timeout for the Worklight Development Server startup. Because the server must start itself and all the applications during this startup period, this default timeout period might not always be long enough when the number of deployed projects is medium to large.

Resolving the problem

To resolve this problem, you can apply one or both of the following workarounds:

- Complete the following steps to increase the default timeout value:
  1. In Worklight Studio, open the **Servers** view.
  2. Double-click **Worklight Development Server** to open the Overview page.
  3. Expand the **Timeouts** section.
  4. In the **Start (in seconds)** field, increase the value. Consider doubling the default value; that is, set it to 120 seconds.
- Complete the following steps to remove the unnecessary applications from the project:
  1. In Worklight Studio, open the **Servers** view.
  2. Right-click **Worklight Development Server**, and then click **Add and Remove**.
  3. In the **Configured** field, click all the applications that you do not intend to work on, and then click **Remove**.

After you have applied one or both of the workarounds, restart the server.

The Build All Environments command

Information about the menu command that is used to build or rebuild an application or an environment, without deploying and running it to a server.

About this task

You use this menu option when you want to build or rebuild your application or environments in Worklight Studio, but not deploy them to the test server. This feature is useful when you are preparing to deploy for QA or Production environments.

Note: This command name is context-sensitive in that it does not always display the name **Build All Environments**. If you right-click the name of one of your environments, the name of that environment is displayed in the command name instead. For example, if you right-click the `android` folder in your project hierarchy, this command appears as **Run As** > **Build Android Environment**. For consistency, when referring to this menu command in this IBM Worklight user documentation, the default name of this command (**Build All Environments**) is used throughout.

Procedure

When you choose this command, Worklight Studio performs the following actions, depending on the project element that is selected:

- If you right-click the main application node and choose **Run As** > **Build All Environments**, Worklight Studio builds your application and all of its included environments.
- If you right-click on only a single environment and choose (for example) **Run As** > **Build iPhone Environment**, Worklight Studio builds only the selected environment.
- No deployment takes place, either to the designated test server or to a different Worklight Server.

Important:

If you checked the **Build the application to work with a different Worklight server** option in the Configure Worklight Build and Deploy Target dialog, then this menu option triggers a build using that Worklight Server information, and recognizes it over the test server setting. If this option is cleared, the build occurs using the build settings for the designated test server.

Any command that triggers the deployment of an application to the Worklight Development Server uses the Worklight Development Server settings to rebuild the application; the configuration settings for the target server are ignored. This is true for the following commands:

- **Run As** > **Run on Worklight Development Server**.
- **Run As** > **Deploy Worklight Adapter** (to deploy the selected adapter).
- **Run As** > **Xcode project** (to build the selected `iphone` or `ipad` environment when working in Worklight Studio on a Mac).

The command **Run As** > **Build All Environments** should be the last command you run before you deploy the project WAR file and other artifacts.

The Preview command

Information about the menu command that is used to preview an application or an environment.

About this task

You use this menu option when you want to preview an application or one of its environments, without triggering a rebuild and redeployment of it to the designated test server.

Note: The Preview feature requires that your designated test server is running and that the application was built at least once for current test server configuration.

Procedure

When you select **Run As** > **Preview**, Worklight Studio displays an instant preview, depending on the project element that is selected:

- **Common preview –** If you right-click the `common` folder in your project hierarchy and choose this command, Worklight Studio previews all common resources. This action opens a browser with a simple preview of your application, independent of any development environment you might have installed.
- **Single environment preview –** If you right-click an environment folder in your project hierarchy and choose this command, Worklight Studio previews that environment with the Mobile Browser Simulator.
- **All environments preview –** If you right-click the main application node in your project hierarchy and choose this command, Worklight Studio previews that environment with the Mobile Browser Simulator, displaying a preview for every environment you added to your application.

Important: If you use an IBM Worklight Shell Component in your inner application, you might see incorrect results with a browser-based preview of the application.

In Worklight Studio V6.1.0, the preview feature that you can start by selecting **Run As** > **Preview** or from the Worklight Console, changed. As an unintended side-effect, the preview of an inner application that references a shell component might not render as expected. Specifically, the fragments that normally get injected into the HTML page for the inner application are missing. As a result, any additional links, such as scripts, or CSS, are omitted during the preview. Also, if an extra user interface is defined in the shell, it is omitted as well.

To see a correct preview for an inner application that uses a shell, build the application and start it by using either a device emulator (Android or iOS) or an actual native device.

The Build Settings and Deploy Target command

Information about the menu command that is used to create build and deploy settings for Worklight applications.

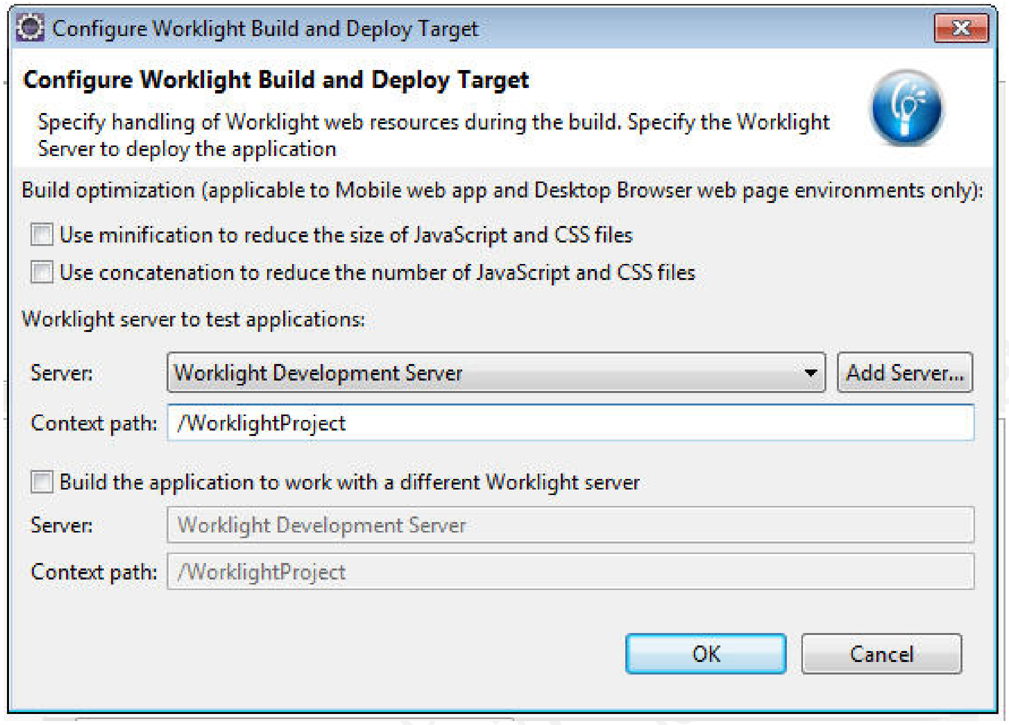
In previous versions of IBM Worklight Studio, you accessed the build and deploy settings for a given Worklight project through a number of different dialogs. Since IBM Worklight 6.1.0, those settings are consolidated in a single dialog. As a result, configuring a project to build and deploy to a local test server or to build for a remote server are available through the same dialog, along with other build settings options.

This new dialog, which is accessed by choosing the **Run As** > **Build Settings and Deploy Target** menu command, takes the place of several commands or dialogs in previous versions of Worklight Studio. It replaces:

- The **Run As** > **Build for Remote Server** menu command and its associated dialog.
- The **Run As** > **Apply Build Settings** menu command and its associated dialog.
- The **Change Target Server** menu command and its associated dialog.

The Build Settings and Deploy Target dialog

When you right-click on an application or an environment and select **Run As** > **Build Settings and Deploy Target**, the following dialog is displayed:



Configure Worklight Build and Deploy Target

Configure Worklight Build and Deploy Target

Specify handling of Worklight web resources during the build. Specify the Worklight Server to deploy the application

Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

□ Use minification to reduce the size of JavaScript and CSS files

□ Use concatenation to reduce the number of JavaScript and CSS files

Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

□ Build the application to work with a different Worklight server

Server: Worklight Development Server

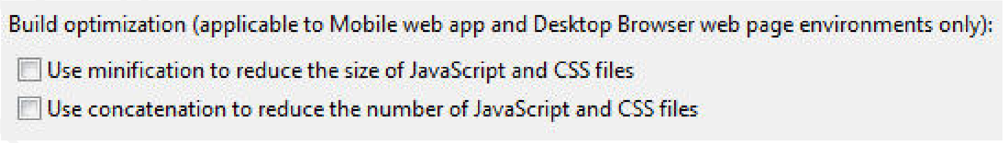
Context path: /WorklightProject

Important: This dialog is used only to specify configurations and settings; clicking **OK** does not trigger a build. Any time that you make a modification with this dialog, you must rebuild your application and environments for your changes to take effect, by using the **Run As** > **Build All Environments** menu command.

The dialog contains three main areas, used to perform different actions. Each of these areas is covered in the sections that follow.

Apply build optimization settings

Build optimizations are useful to reduce the size of an application, improve its performance, or reduce its load time. The available optimizations, minification and concatenation, are disabled by default for every project, but you can enable them by checking the appropriate option in the following screen capture.



Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

□ Use minification to reduce the size of JavaScript and CSS files

□ Use concatenation to reduce the number of JavaScript and CSS files

You use this area of the dialog if you want to change the build settings for Desktop Browser and Mobile Web environments for the currently selected server, and want to apply these new minification and concatenation settings to future builds.

Important: The build optimization settings that you choose apply to the Worklight Server selected in the rest of the dialog. That is, if **Build the application to work with a different server** is selected, these optimization settings apply to that server when you click **OK**. If **Build the application to work with a different server** is not selected, they apply to the test server selected in the **Server** field of the **Worklight server to test applications** area.

For more information about build settings, see “IBM Worklight application build settings” on page 516.

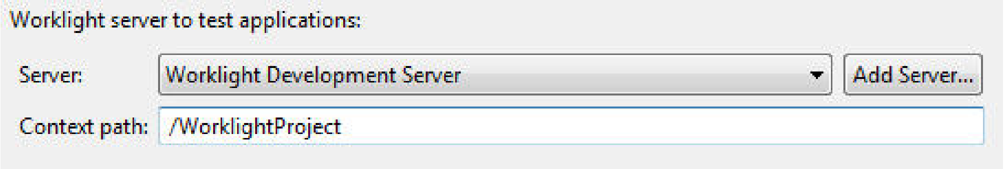
For more information about minification, see “Minification of JS and CSS files” on page 519.

For more information about concatenation, see “Concatenation of JS and CSS files” on page 521.

Worklight Server to test applications

In this area of the dialog, you can set the Worklight Server that you want to use to test your applications and environments. The default setting is **Worklight Development Server**. This name refers to the embedded instance of WebSphere Application Server Liberty Profile and Worklight Server that is created when you install Worklight Studio.

But you can have other test servers that you want to use. For example, you can have a local or shared instance of Worklight Server running on WebSphere Application Server Liberty Profile or Apache Tomcat. Using this area of the dialog, you can choose which of these servers you want to use as a default for testing during development.



Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

The **Server** field enables you to select from a list of configured test servers in your Worklight Studio development environment. You can also add a test server by clicking **Add Server**.

The **Context path** field enables you to specify the web application context path to be used when you deploy and run on the selected test server. By default this field is set to `/<your_project_name>`.

Note: If you use this area of the Configure Worklight Build and Deploy Target dialog to choose a different test server than the internal Worklight Development Server, the name of the server that is selected in the **Server** field is displayed in the command name instead. For example, if you add a local instance of Worklight Server running on Apache Tomcat, name it "Tomcat Test Server" and designate it as your default test server, this command appears in the menus as **Run As** > **Run on Tomcat Test Server**.

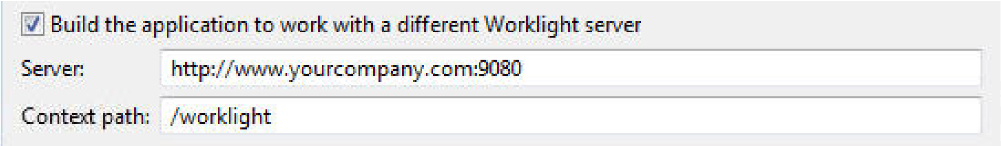
Important: When you change your designated test server, the new server remains the default for the **Run As** > **Run on <name of test server>** command until you change it again. All subsequent builds created with this command are deployed to and run on the test server you selected.

Build the application to work with a different Worklight Server

You use this option when you want to build your project in Worklight Studio and run it on another instance of Worklight Server that is running externally to your Eclipse development environment. For example, after you test locally, you use this area of the dialog to build your application for deployment to a production server.

Note: Both Worklight Studio Consumer Edition and Worklight Studio Enterprise Edition provide the capability to deploy to the internal Worklight Development Server, and, using this area of the dialog, to a remote server. The Worklight Studio Developer Edition is provided for evaluation purposes, and you can deploy only to the internal Worklight Development Server.

This area of the Configure Worklight Build and Deploy Target dialog becomes active when you select **Build the application to work with a different server**:



☑ Build the application to work with a different Worklight server

Server: http://www.yourcompany.com:9080

Context path: /worklight

The **Server** field is required, and contains the URL for the remote target server. The entry must use the format: `http(s)://`*`<hostname>`*`:`*`<port>`*.

The **Context path** field specifies the web application context path to be used when deploying to this server.

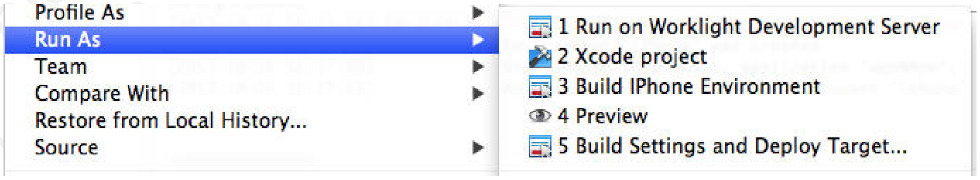
To deploy the resulting WAR file and other artifacts, you must use the Worklight Console, the supplied Ant tasks, or the Worklight Server Configuration Tool, following the procedures that are listed in "Deploying the project WAR file" on page 714.

Additional Run As menu options

Information about additional menu commands that can appear in your **Run As** menus, depending on platform and the external development environments you have installed.

There are a number of commands that can appear on the **Run As** menus, depending on context-sensitivity, the computer platform you are working on, and which external development environments you have installed on your computer.

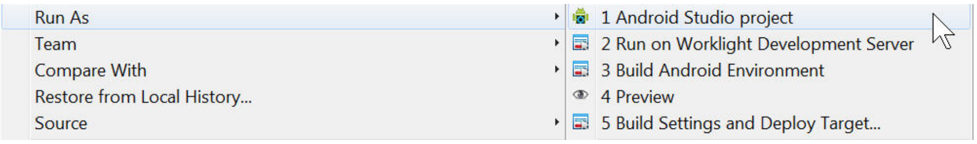
- If you are working in Worklight Studio on a Mac, you can right-click the `iphone` or `ipad` environment and use the **Run As** > **Xcode project** option to build the environment and open the Xcode development environment for that project.



Profile As
Run As
Team
Compare With
Restore from Local History...
Source
1 Run on Worklight Development Server
2 Xcode project
3 Build IPhone Environment
4 Preview
5 Build Settings and Deploy Target...

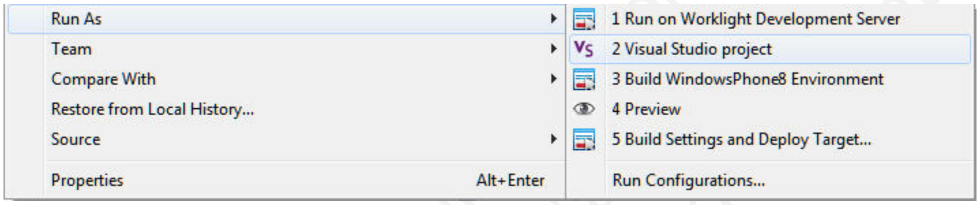
For more information about the iOS development environment, see the module *Previewing your application on iOS*, under category 2, *Hello Worklight*, in Chapter 3, "Tutorials and samples," on page 27.

- If you are working in Worklight Studio on a computer with the ADT plugin (Android Development Tools for Eclipse) installed, you can right-click the `android` environment and use the **Run As** > **Android Studio project** option to open that project in the Android development environment.


Run As
Team
Compare With
Restore from Local History...
Source
1 Android Studio project
2 Run on Worklight Development Server
3 Build Android Environment
4 Preview
5 Build Settings and Deploy Target...

For more information about the Android development environment, see the module *Previewing your application on Android*, under category 2, *Hello Worklight*, in Chapter 3, "Tutorials and samples," on page 27.

- If you are working in Worklight Studio on a computer running Windows 8, with Microsoft Visual Studio installed, you can right-click the `windowsphone8` or `windows8` environment and use the **Run As** > **Visual Studio project** option to open that project in the Visual Studio development environment.


Run As
Team
Compare With
Restore from Local History...
Source
Properties
Alt+Enter
1 Run on Worklight Development Server
2 Visual Studio project
3 Build WindowsPhone8 Environment
4 Preview
5 Build Settings and Deploy Target...
Run Configurations...

For more information about the Windows Phone 8 development environment, see the module *Previewing your application on Windows Phone 8*, under category 2, *Hello Worklight*, in Chapter 3, "Tutorials and samples," on page 27.

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 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.

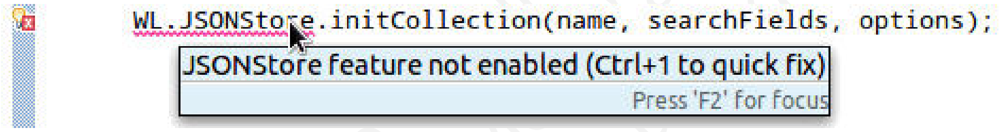
With IBM Worklight, you can include or exclude features from the application build if those features are not required. For example, JSONStore offers many benefits, if code that references it is actually used in the application. If it is not used, the JSONStore resources greatly increase the application 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 platformVersion="6.1.0.00.20131126-0630" id="myApp" xmlns="http://www.worklight.com/app
    ...
    <features>
        <JSONStore/>
    </features>
    ...
</application>
```

For more information about Application Descriptor attributes, see “The application descriptor” on page 331.

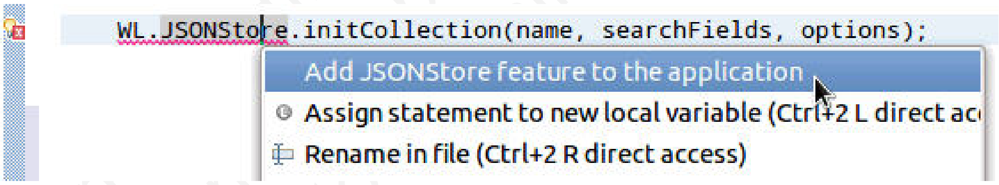
When you first create an IBM Worklight application, the <features> tag is automatically created in the application-descriptor.xml file, with no contents. What this means is that if you use JSONStore in your code, it is not automatically added to the builds. When you run the application, you receive an error, as shown in the following screen capture:



```
WL.JSONStore.initCollection(name, searchFields, options);
```

JSONStore feature not enabled (Ctrl+1 to quick fix)

You can resolve this situation by using an Eclipse QuickFix:



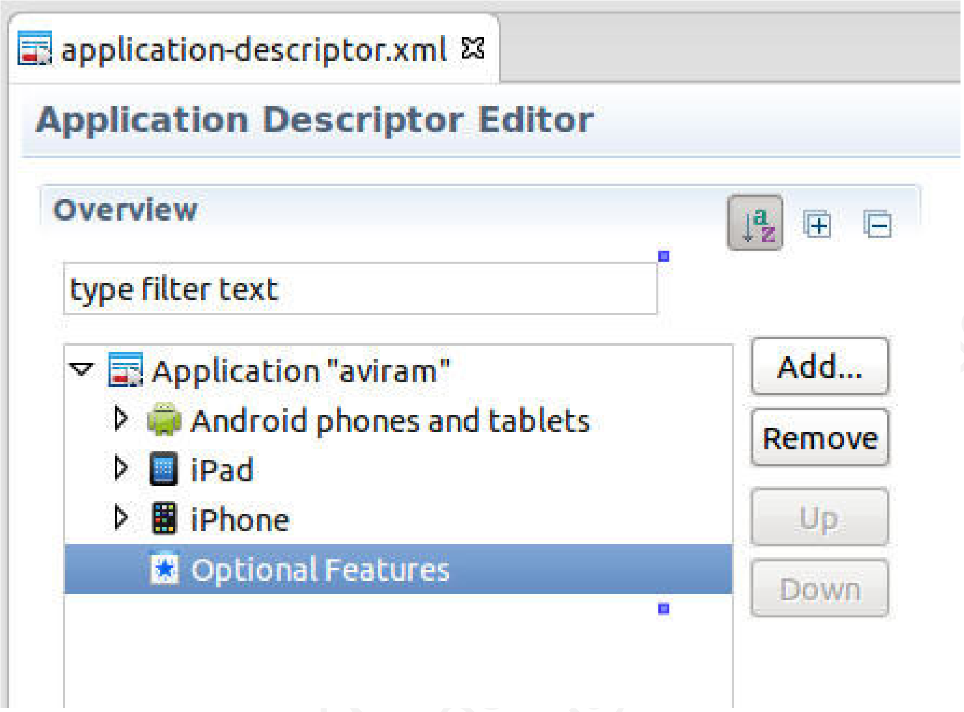
```
WL.JSONStore.initCollection(name, searchFields, options);
```

Add JSONStore feature to the application
Assign statement to new local variable (Ctrl+2 L direct ac
Rename in file (Ctrl+2 R direct access)

But you can also choose which features to include in the build with the Worklight Studio editor, as shown in the following procedure.

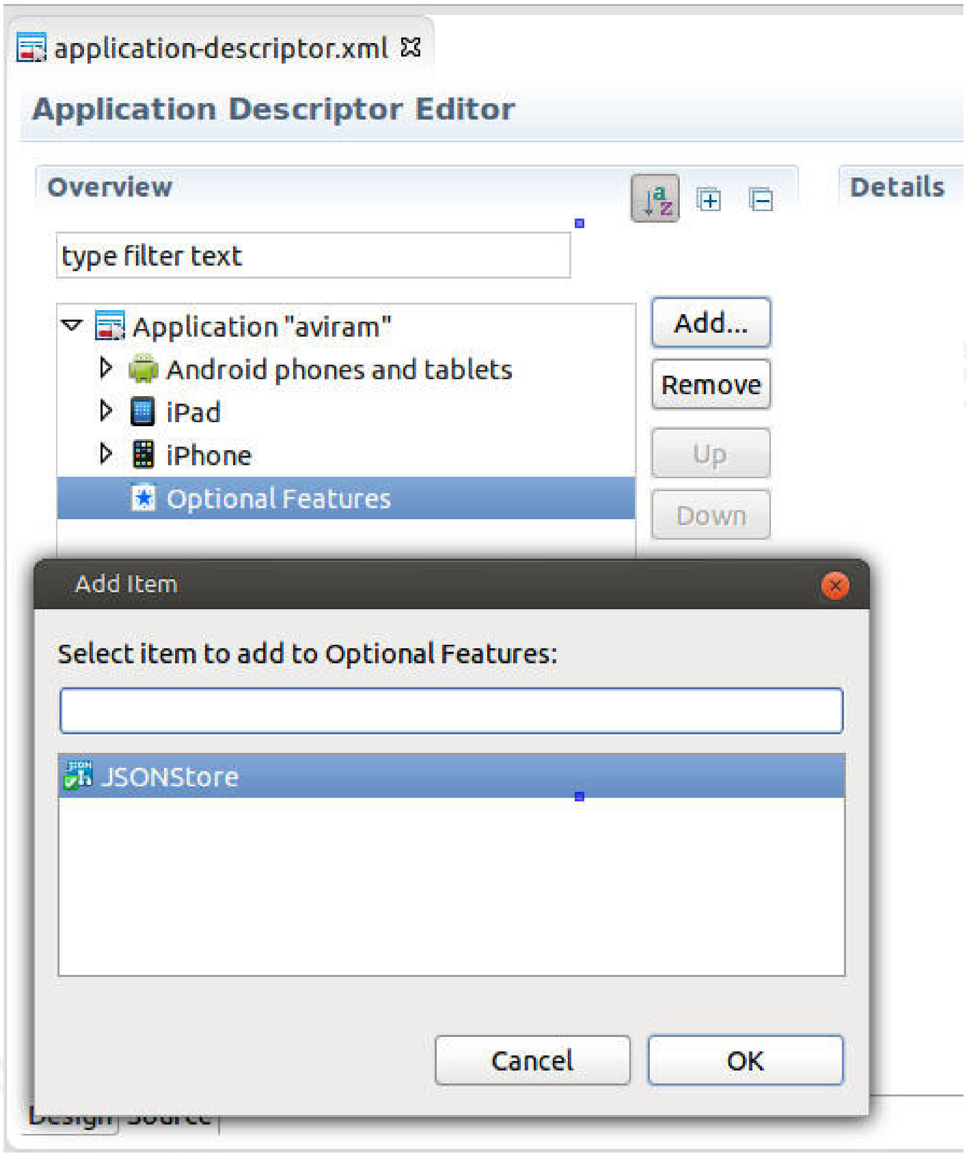
To include or exclude features in Worklight Studio

1. In Worklight Studio, open the application-descriptor.xml file for your application with the Application Descriptor Editor:

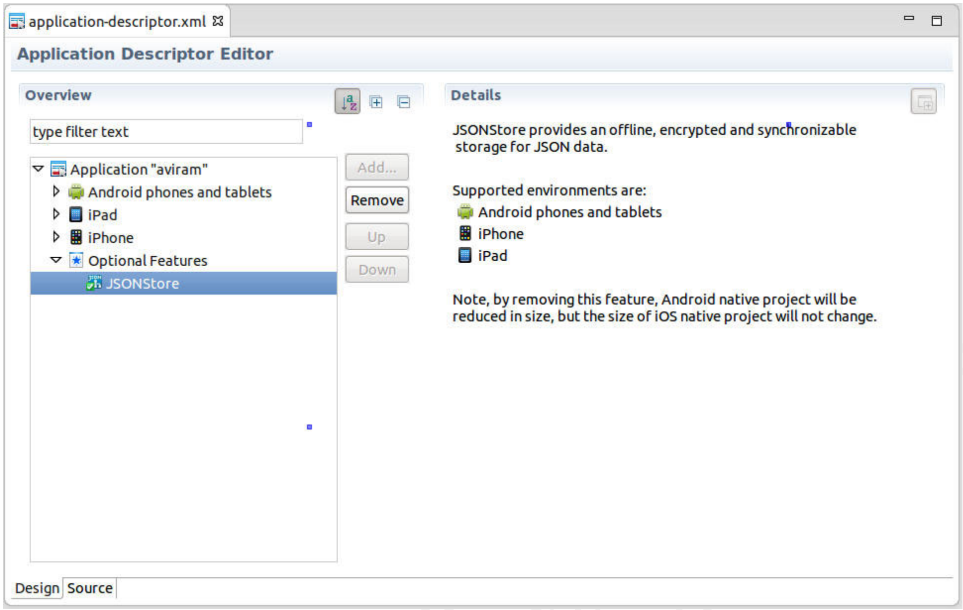


If the **Optional Features** element is empty (as in the screen capture), no features such as JSONStore are included in the build.

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:



JSONStore provides an offline, encrypted and synchronizable storage for JSON data.

Supported environments are:
Android phones and tablets
iPhone
iPad

Note, by removing this feature, Android native project will be reduced in size, but the size of iOS native project will not change.

5. 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.

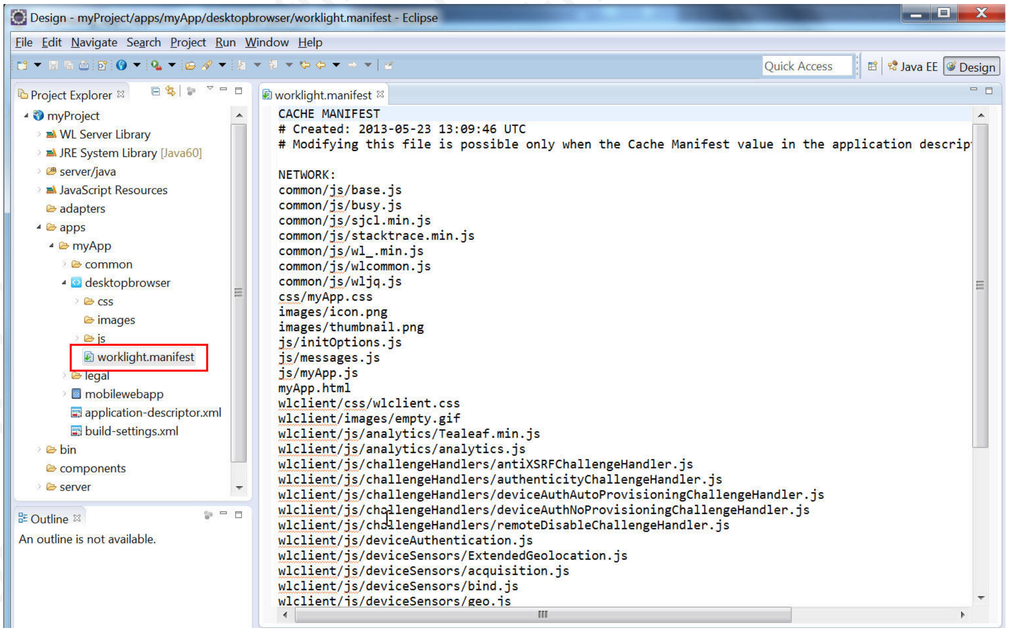
If unnecessary or redundant files are included, they must all be fetched before the application can start, which can create a poor user experience. The procedure that follows documents ways in which you can edit the Cache Manifest to reduce the start time for your Desktop Browser and Mobile Web applications.

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.

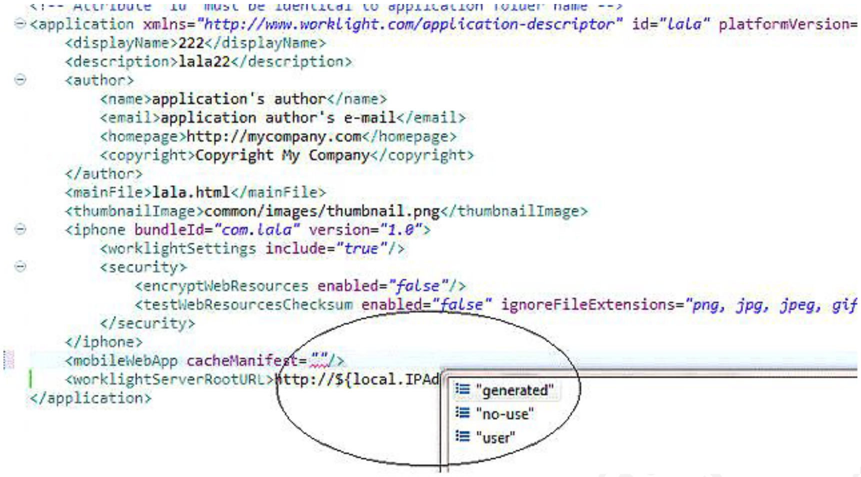
With IBM Worklight, you can control the Cache Manifest in web environments (Desktop Browser and Mobile Web). The name of the Cache Manifest file is `worklight.manifest`. This file is located in the folder for each of these types of environments.

You can now view (and edit) the contents of this file in Worklight Studio, as shown in the following screen capture:



```
CACHE MANIFEST
# Created: 2013-05-23 13:09:46 UTC
# Modifying this file is possible only when the Cache Manifest value in the application descrip

NETWORK:
common/js/base.js
common/js/busy.js
common/js/sjcl.min.js
common/js/stacktrace.min.js
common/js/wl_.min.js
common/js/wlcommon.js
common/js/wljq.js
css/myApp.css
images/icon.png
images/thumbnail.png
js/initOptions.js
js/messages.js
js/myApp.js
myApp.html
wlclient/css/wlclient.css
wlclient/images/empty.gif
wlclient/js/analytics/Tealeaf.min.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/ExtendedGeolocation.js
wlclient/js/deviceSensors/acquisition.js
wlclient/js/deviceSensors/bind.js
wlclient/js/deviceSensors/geo.js
```

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 cacheManifest=""/>
    <worklightServerRootURL>http://${local.IPAd
</application>
```

For more information about Application Descriptor attributes, see “The application descriptor” on page 331.

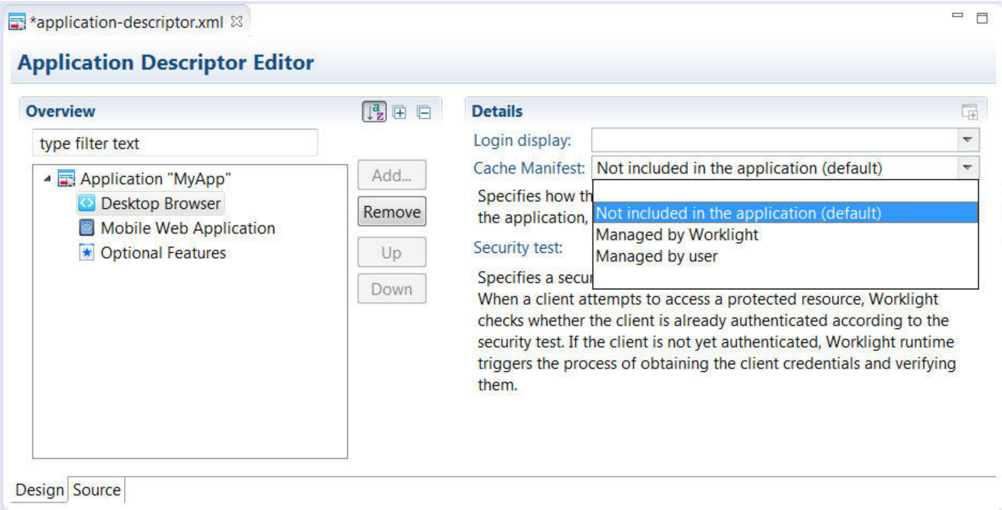
The <cacheManifest> attribute accepts three values, as shown in the following table. No matter which value or mode is selected, if the Cache Manifest does not exist, the Worklight Studio builder generates the default Cache Manifest to give you something to start with. But after creating this file, the builder leaves the resulting Cache Manifest file in its default **no-use** mode unless you explicitly change the setting.

Table 75. <cacheManifest> properties

Property	Description
generated	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: • For Desktop Browser environments – all resources are under NETWORK, which means: no cache at all. • For Mobile Web environments – all resources are under CACHE, which means: cache everything. In **generated** 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.
no-use	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.
user	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:



Login display:

Cache Manifest: Not included in the application (default)

Specifies how th
the application,

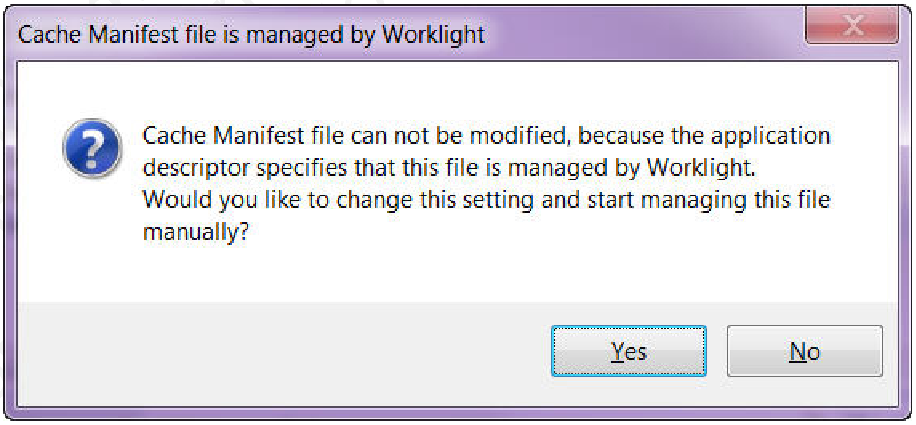
Security test:

Specifies a secu
When a client attempts to access a protected resource, Worklight checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, Worklight runtime triggers the process of obtaining the client credentials and verifying them.

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:



Cache Manifest file is managed by Worklight

Cache Manifest file can not be modified, because the application descriptor specifies that this file is managed by Worklight.
Would you like to change this setting and start managing this file manually?

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.

Since IBM Worklight V6.0.0, a file named `build-settings.xml` is created when a new Worklight application is created, 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 76. Options for the <minification> `level` attribute

Value	Description
none	No minification is done on your code by the Worklight Studio builder.
whitespaces	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 **Remove whitespaces and comments**.)

`simple`	Removes the same white space and comments as **`whitespaces`**, 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 **`simple`** 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 `eval()` statements. (In the Worklight Studio Build Settings Editor, this attribute is called **Google Closure Compiler Simple Optimization**.)

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 519.

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.

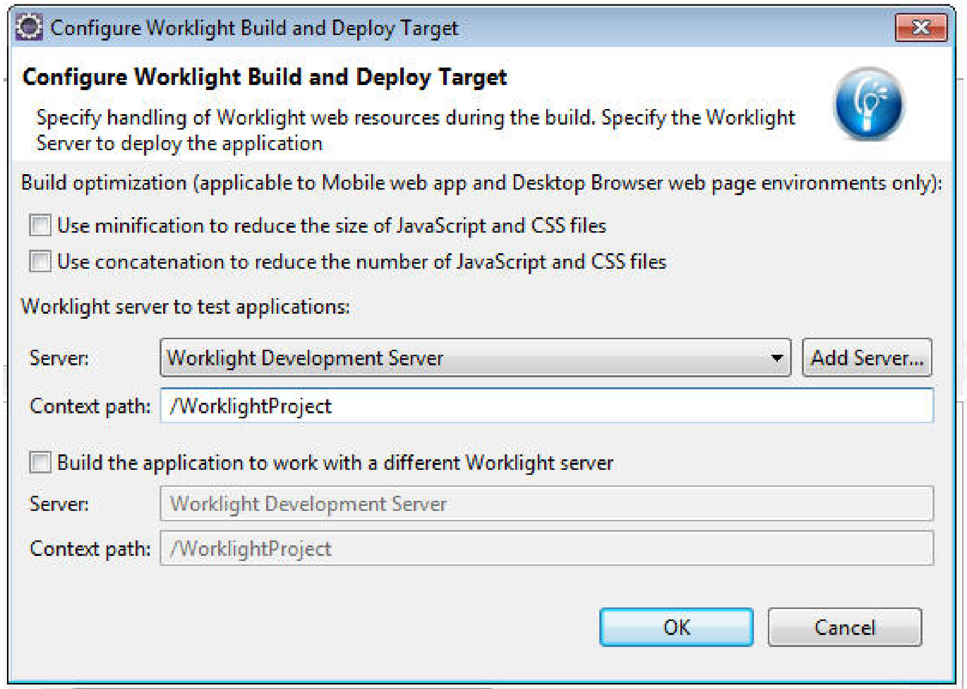
For more information about concatenation,see “Concatenation of JS and CSS files” on page 521.

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 (or the main application node) and choose **Run As** > **Build Settings and Deploy Target** from the menu.

   The Build Settings and Deploy Target window is displayed:


Configure Worklight Build and Deploy Target

Configure Worklight Build and Deploy Target

Specify handling of Worklight web resources during the build. Specify the Worklight Server to deploy the application

Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

□ Use minification to reduce the size of JavaScript and CSS files

□ Use concatenation to reduce the number of JavaScript and CSS files

Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

□ Build the application to work with a different Worklight server

Server: Worklight Development Server

Context path: /WorklightProject

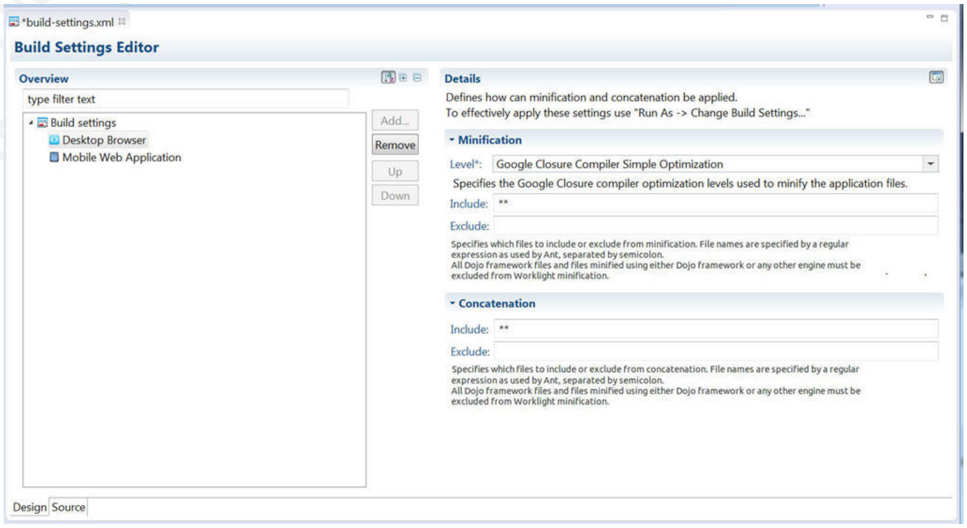
2. In the **Build optimization** area of the dialog, select the check box of the feature or features you want to use when you build this environment.
3. Click **OK**.

Note: This action does not trigger an automatic build. To build or rebuild using these new settings, you must use either the **Run As** > **Run on Worklight Development Server** or the **Run As** > **Build...** menu commands.

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:


*build-settings.xml

Build Settings Editor

Overview

type filter text

- Build settings
  - Desktop Browser
  - Mobile Web Application

Details

Defines how can minification and concatenation be applied.
To effectively apply these settings use "Run As -> Change Build Settings..."

Minification

Level*: Google Closure Compiler Simple Optimization

Specifies the Google Closure compiler optimization levels used to minify the application files.

Include: **

Exclude:

Specifies which files to include or exclude from minification. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

Concatenation

Include: **

Exclude:

Specifies which files to include or exclude from concatenation. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

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 516 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:

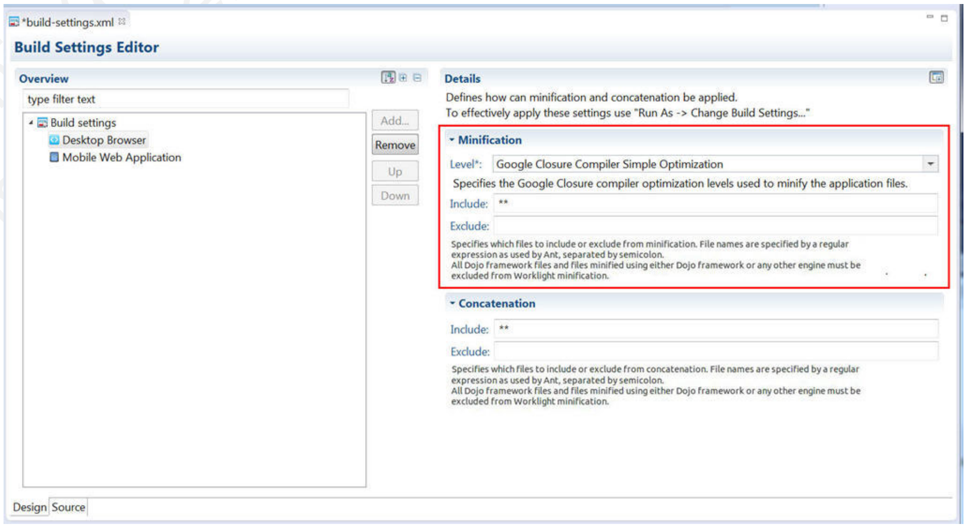
Table 77. Options for the <minification> `level` attribute

Value	Description
`none`	No minification is done on your code by the Worklight Studio builder.
`whitespaces`	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 **Remove whitespaces and comments**.)
`simple`	Removes the same white space and comments as **`whitespaces`**, 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 **`simple`** 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 `eval()` statements. (In the Worklight Studio Build Settings Editor, this attribute is called **Google Closure Compiler Simple Optimization**.)

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:



Defines how can minification and concatenation be applied.
To effectively apply these settings use "Run As -> Change Build Settings..."

Minification

Level*: Google Closure Compiler Simple Optimization

Specifies the Google Closure compiler optimization levels used to minify the application files.

Include: **

Exclude:

Specifies which files to include or exclude from minification. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

Concatenation

Include: **

Exclude:

Specifies which files to include or exclude from concatenation. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

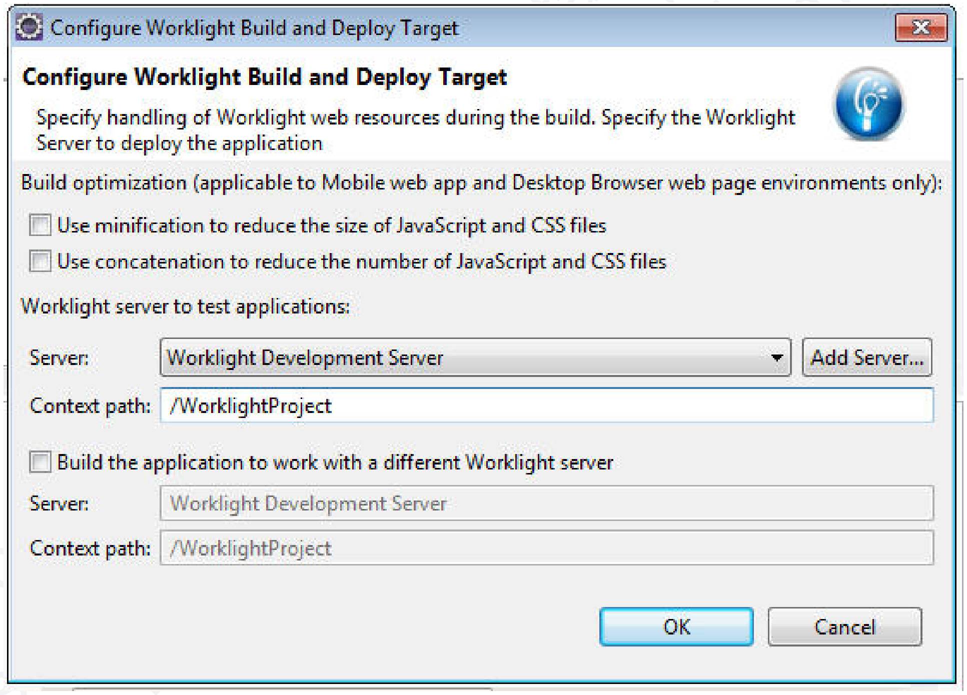
2. Select the wanted minification level from the **Level** field:
   - **None (Default)** specifies the **`none`** attribute in the table previously mentioned.

- **Remove whitespaces and comments** specifies the **whitespaces** attribute in the table previously mentioned.
- **Google Closure Compiler Simple Optimization** specifies the **simple** attribute in the table previously mentioned.

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 516.
4. To instruct Worklight Studio to use concatenation during the build, in Worklight Studio, right-click the **desktopbrowser** or **mobilewebapp** element of your application (or the main application node) and choose **Run As** > **Build Settings and Deploy Target** from the menu.

   The Build Settings and Deploy Target window is displayed:



Configure Worklight Build and Deploy Target

Configure Worklight Build and Deploy Target

Specify handling of Worklight web resources during the build. Specify the Worklight Server to deploy the application

Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

☐ Use minification to reduce the size of JavaScript and CSS files

☐ Use concatenation to reduce the number of JavaScript and CSS files

Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

☐ Build the application to work with a different Worklight server

Server: Worklight Development Server

Context path: /WorklightProject

5. In the **Build optimization** area of the dialog, select **Use minification to reduce the size of JavaScript and CSS files**.
6. Click **OK**.
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 516 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.

Since 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).

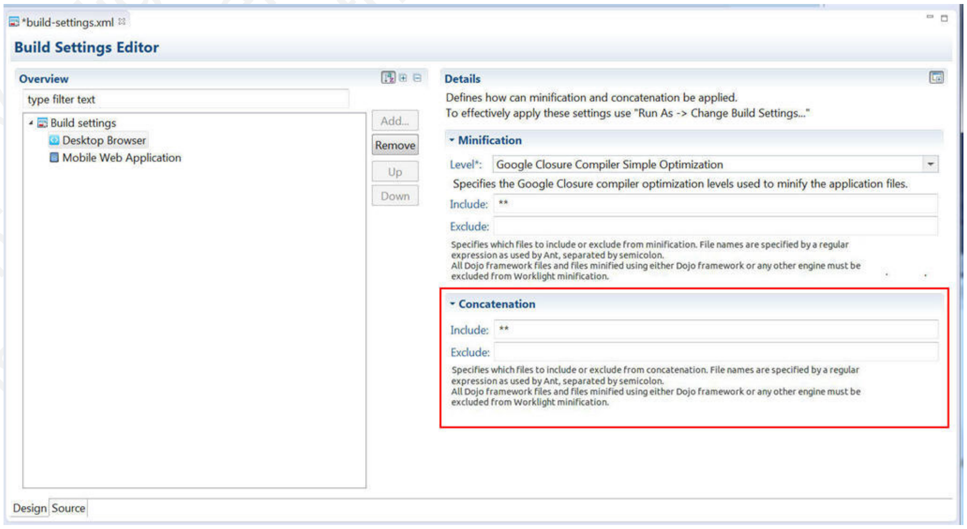
During concatenation, several resources (for example, JavaScript files and inline scripts) are copied into a new file, which is then referenced by the application HTML. References to the original resources are removed from the HTML. This means that less communication between the device and web server 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:



Defines how can minification and concatenation be applied.
To effectively apply these settings use "Run As -> Change Build Settings..."

Level*: Google Closure Compiler Simple Optimization

Specifies the Google Closure compiler optimization levels used to minify the application files.

Include: **

Exclude:

Specifies which files to include or exclude from minification. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

Include: **

Exclude:

Specifies which files to include or exclude from concatenation. File names are specified by a regular expression as used by Ant, separated by semicolon.
All Dojo framework files and files minified using either Dojo framework or any other engine must be excluded from Worklight minification.

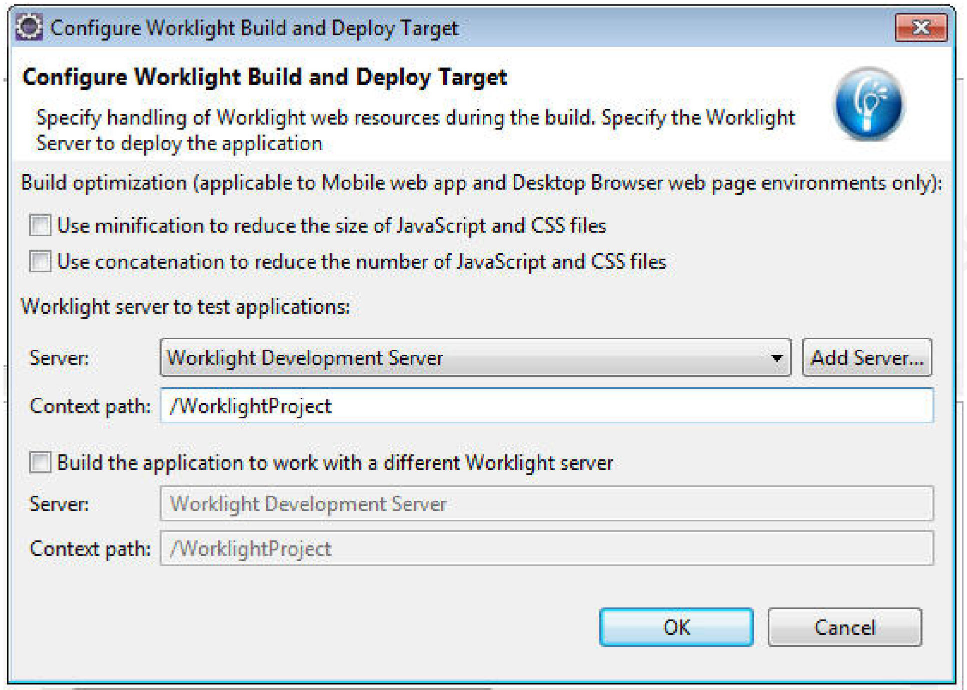
2. Enter the list of files to be concatenated or not concatenated in the **includes** and **excludes** fields. When you save, these settings become part of the application code.

   Use the Ant syntax that is described in the following “Syntax and examples” on page 523 section and in “IBM Worklight application build settings” on page 516.

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 (or the main application node) and choose **Run As** > **Build Settings and Deploy Target** from the menu.

The Build Settings and Deploy Target window is displayed:



Configure Worklight Build and Deploy Target

Configure Worklight Build and Deploy Target

Specify handling of Worklight web resources during the build. Specify the Worklight Server to deploy the application

Build optimization (applicable to Mobile web app and Desktop Browser web page environments only):

☐ Use minification to reduce the size of JavaScript and CSS files

☐ Use concatenation to reduce the number of JavaScript and CSS files

Worklight server to test applications:

Server: Worklight Development Server

Context path: /WorklightProject

☐ Build the application to work with a different Worklight server

Server: Worklight Development Server

Context path: /WorklightProject

4. In the **Build optimization** area of the dialog, select **Use concatenation to reduce the number of JavaScript and CSS files**.
5. Click **OK**.
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/main.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/main.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.jsand 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, main.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.

With IBM Worklight, you can create and configure adapters manually, or you can also automatically generate SAP Netweaver Gateway or SOAP adapters with the services discovery wizard. For more information about how to automatically generate adapters, see “Generating adapters with the services discovery wizard” on page 547.

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.

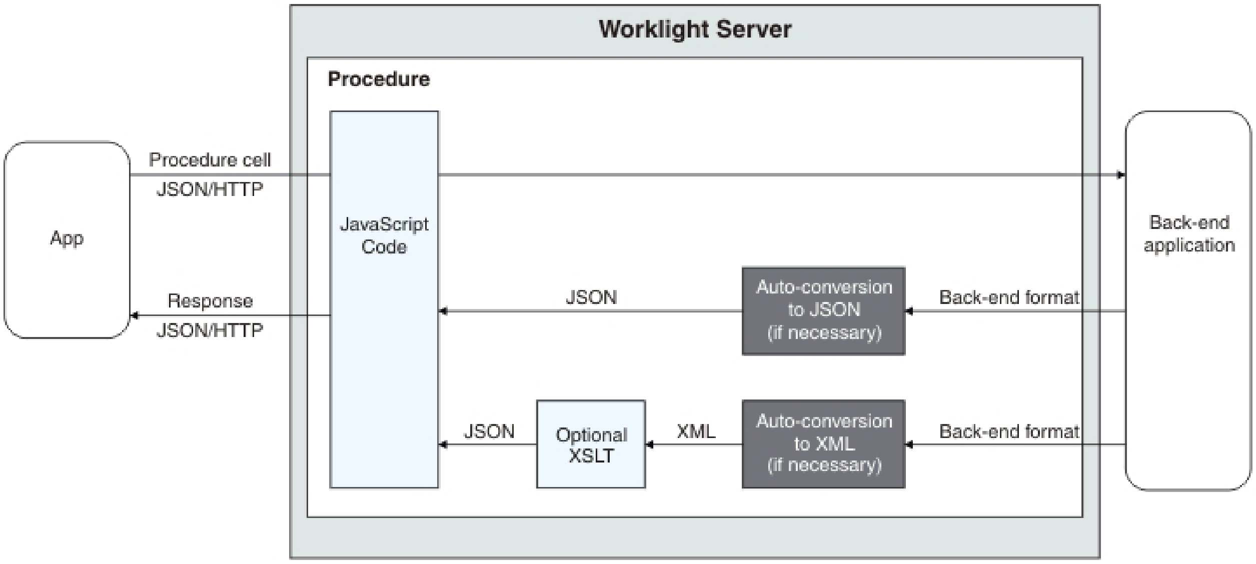


Figure 64. The adapter framework

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 778.
- 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 534.

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 the `createEventSource` method in the WL.Server class.

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 78. <adapter> element attributes

Attribute	Description
name	Mandatory. The name of the adapter. This name must be unique within the Worklight Server. It can contain alphanumeric characters and underscores, and must start with a letter. **Note:** After an adapter has been defined and deployed, its name cannot be modified.
xmlns:*namespace*	Mandatory. Defines schema namespaces. This attribute must appear three times, as follows: `xmlns:xsi` – Defines the namespace associated with the `http://www.w3.org/2001/XMLSchema-instance` schema. `xmlns:wl` – Defines the namespace associated with the `http://www.worklight.com/integration` schema. `xmlns:`*`namespace`* – Defines the namespace associated with the schema related to the back-end application, for example, `xmlns:sap` or `xmlns:sql`.
xsi:schemaLocation	Optional. Identifies the schema locations, in the following format: `xsi:schemaLocation="http://www.worklight.com/integration l` 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 79. <adapter> element subelements

Subelement	Description
<displayName>	**Note**: 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."
<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 532.

<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 80. <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 534
The structure of the `<ConnectionPolicy>` element.

"The <connectionPolicy> element of the SQL adapter" on page 537
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 540
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 81. <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.
connectAs	Optional. Defines how to create a connection to the back end for invoking the retrieve procedure. Valid values are as follows: `server`: Default. The connection to the back end is created according to the connection policy defined for the adapter. `endUser`: The connection to the back end is created with the user's identity. Only valid if a user realm has been identified in the security tests for this procedure.
requestTimeoutInSeconds	Optional. The timeout in seconds for waiting until receiving a response from the back end, including the time for opening the connection. The default is 30 seconds.
audit	Optional. Defines whether calls to the procedure are logged in the audit log. The log file is *Worklight Project Name*/server/log/audit/audit.log. Valid values are as follows: `true`: Calls to the procedure are logged in the audit log. `false`: Default. Calls to the procedure are not logged in the audit log.
securityTest	Optional. The name of the security test that you want to use to protect the adapter procedure, as defined in the `authenticationConfig.xml` file.

The root element of the HTTP adapter XML file

The structure of the root element.

The root element of the HTTP adapter has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter>
name="adapter-name"
authenticationRealm="realm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:wl="http://www.worklight.com/integration"
xmlns:http="http://www.worklight.com/integration/http"
```

```
xsi:schemaLocation=
"http://www.worklight.com/integration integration.xsd
http://www.worklight.com/integration/http http.xsd">
...
</wl:adapter>
```

The <connectionPolicy> element of the HTTP adapter

The structure of the <ConnectionPolicy> element.

The <ConnectionPolicy> element has the following structure:

```
<connectionPolicy
  xsi:type="http:HTTPConnectionPolicyType"
  cookiePolicy="cookie-policy"
  maxRedirects="int">
  <protocol>protocol</protocol>
  <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 82. <ConnectionPolicy> element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `http:HTTPConnectionPolicyType`.
cookiePolicy	Optional. This attribute sets how the HTTP adapter handles cookies that arrive from the back-end application. Valid values are as follows: • RFC_2109 (The default) • RFC_2965 • NETSCAPE • IGNORE_COOKIES
maxRedirects	Optional. The maximum number of redirects that the HTTP adapter can follow. This attribute is useful when the back-end application sends circular redirects as a result of some error, such as authentication failures. In IBM Worklight V6.1.0, starting with Fix Pack 2, if the attribute is set to 0, the adapter does not attempt to follow redirects at all, and the HTTP 302 response is returned to the user. Moreover, the default value is 10, instead of 20 previously.

The <ConnectionPolicy> element has the following subelements:

Table 83. `<ConnectionPolicy>` element subelements

Subelement	Description
protocol	Optional. The URL protocol to use. Possible values are `http` (default) and `https`.
domain	Mandatory. The host address.

Table 83. <ConnectionPolicy> element subelements (continued)

Subelement	Description
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 778
sslCertificatePassword	The password of the adapter private SSL key, which is used by the HTTP adapter key manager to access the correct SSL certificate in the keystore. Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication. The keystore setup process is described in "SSL certificate keystore setup" on page 778
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 536.

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 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 535.

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 84. `<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 85. `<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 784.

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 732.

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 86. <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 87. <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 784.

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 88. `<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 89. `<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 90. <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 91. `<connectionPolicy>` element subelements

Subelement	**Description**
namingConnection	Optional. Describes how to connect to an external JNDI repository. Only used if the JNDI objects are not stored in the JEE server that the adapter is deployed in. See “The <namingConnection> element of the JMS adapter” on page 541.

Table 91. <connectionPolicy> element subelements (continued)

Subelement	Description
jmsConnection	Mandatory. Describes the connection factory and optional security details used to connect to the messaging system. See "The <jmsConnection> element of the JMS adapter."

The <namingConnection> element of the JMS adapter

Use the <namingConnection> element to identify how the Worklight Server connects to an external repository.

The JMS Adapter uses administered objects that must be predefined in a JNDI repository. The repository can either be defined in the JEE server context or an external JNDI repository. If you use an external repository, specify the <namingConnection> element to identify how the Worklight server connects to the repository.

The <namingConnection> element has the following attributes:

Attribute	Description
url	Mandatory. The url of the external JNDI repository. For example: *iiop://localhost*. The url syntax is dependent on the JNDI provider.
initialContextFactory	Mandatory. The initialContextFactory class name of the JNDI provider. For example: *com.ibm.Websphere.naming.WsnInitialContextFactory*. The driver, and any associated files, must be placed in the /server/lib directory. If you develop in the Eclipse environment, the driver and associated files must be placed in the /lib directory. **Note:** If you develop for WebSphere Application Server with WebSphere MQ, do not add the WebSphere MQ Java archive (JAR) files to the /lib directory. If the WebSphere MQ JAR files are added, classloading problems will occur because the files already exist in the WebSphere Application Server environment.
user	Optional. User name of a user with authority to connect to the JNDI repository. If user is not specified, the default user name is *guest*.
password	Optional. Password for the user specified in the user attribute. If user is not specified, the default password is *guest*.

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. **Note:** If you are deploying in WebSphere Application Server, the connection factory must be a global JNDI object. The object must be addressed without the `java:comp/env` context. For example: `jms/MyConnFactory` and not `java:comp/env/jms/MyConnFactory`. However, if you are deploying in Tomcat, the connection factory must be addressed including the `java:/comp/env` context. For example: `java:comp/env/jms/MyConnFactory`.
user	Optional. User name of a user with authority to connect to the messaging system.
password	Optional. Password for the user specified in the `user` attribute.

The root element of the SAP Netweaver Gateway adapter XML file

The structure of the root element of the SAP Netweaver Gateway adapter.

The root element of the SAP Netweaver Gateway 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/nwgateway">
  ...
</wl:adapter>
```

The <connectionPolicy> element of the SAP Netweaver Gateway adapter

The structure of the `<connectionPolicy>` element.

The `<connectionPolicy>` element has the following structure:

```
<connectionPolicy xsi:type="nwgateway:NWGatewayHTTPConnectionPolicyType">
  <protocol>protocol</protocol>
  <domain>host-name</domain>
  <port>host-port</port>
  <client>sap-client</client>
  <username>sap-username</username>
  <password>sap-password</password>
  <serviceRootUrl>service-root-url</serviceRootUrl>
  <authentication>...</authentication>
  <sslCertificateAlias>ssl-certificate-alias</sslCertificateAlias>
  <sslCertificatePassword>ssl-certificate-password</sslCertificatePassword>
  <proxy>...</proxy>
</connectionPolicy>
```

The `<connectionPolicy>` element has the following attributes:

Table 92. <connectionPolicy> element attributes

Attribute	Description
xsi:type	Mandatory. The value of this attribute must be set to `nwgateway:NWGatewayHTTPConnectionPolicyType`.

The `<connectionPolicy>` element has the following subelements:

Table 93. `<connectionPolicy>` element subelements

Subelement	Description
protocol	Mandatory. The URL protocol to use. Possible values are `http` (default) and `https`.
domain	Mandatory. The SAP Netweaver Gateway server address.
port	Mandatory. The SAP Netweaver Gateway server port address. The default value is 80.
client	Mandatory. The SAP-Client to be used to contact the Netweaver Gateway. The default value is 1.
username	Mandatory. The user name for contacting the Netweaver Gateway. The default value is sap-username.
password	Mandatory. The password for contacting the Netweaver Gateway. The default value is sap-password.
serviceRootUrl	Mandatory. The root URL for the SAP Netweaver gateway service that you are trying to access.

authentication	Mandatory. Authentication configuration for the SAP Netweaver gateway adapter. A sample authentication follows:
	`<authentication>`
	`  <basic />`
	`  <serverIdentity>`
	`    <client>001</client>`
	`    <username>mygatewayuser</username>`
	`    <password>mygatewaypassword</password>`
	`  </serverIdentity>`
	`</authentication>`
	For more information, see "The <authentication> element of the HTTP adapter" on page 535. The SAP Netweaver Gateway adapter shares the same authentication configuration stanza with the HTTP adapter except that `<serverIdentity>` requires one additional `<client>` tag.

Table 93. <connectionPolicy> element subelements (continued)

Subelement	Description
sslCertificateAlias	The alias of the adapter private SSL key. Used by the SAP Netweaver gateway 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 778.
sslCertificatePassword	The password of the adapter private SSL key. Used by the SAP Netweaver gateway 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 778.
proxy	Optional. Used when the backend application must be accessed through a proxy. For more information, see “The <proxy> element of the HTTP adapter” on page 536. The SAP Netweaver Gateway adapter shares the same proxy configuration stanza with the HTTP adapter.

Creating an IBM Worklight adapter

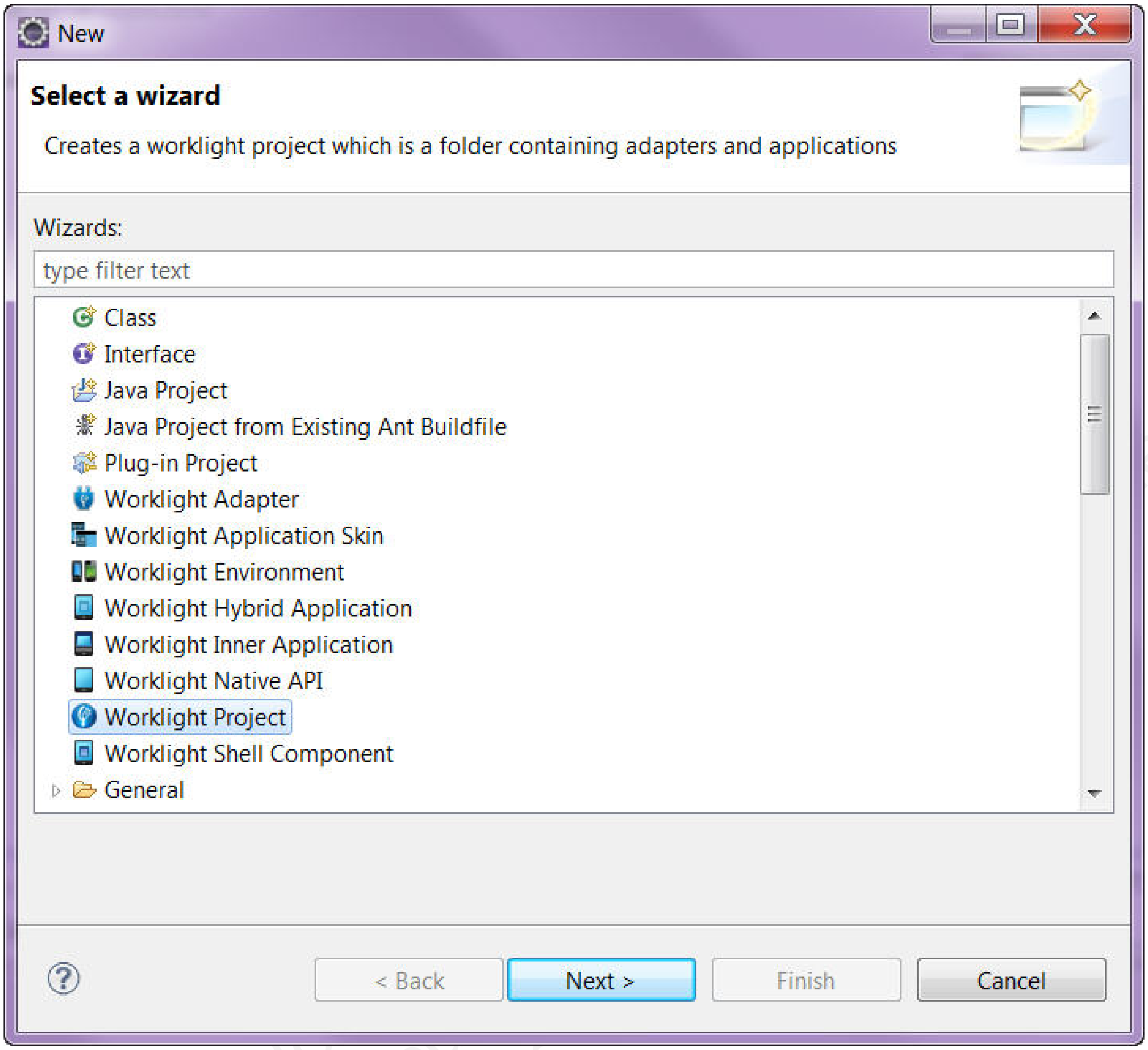
Follow these instructions to create an IBM Worklight project and configure a new IBM Worklight adapter.

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**.



New

Select a wizard

Creates a worklight project which is a folder containing adapters and applications

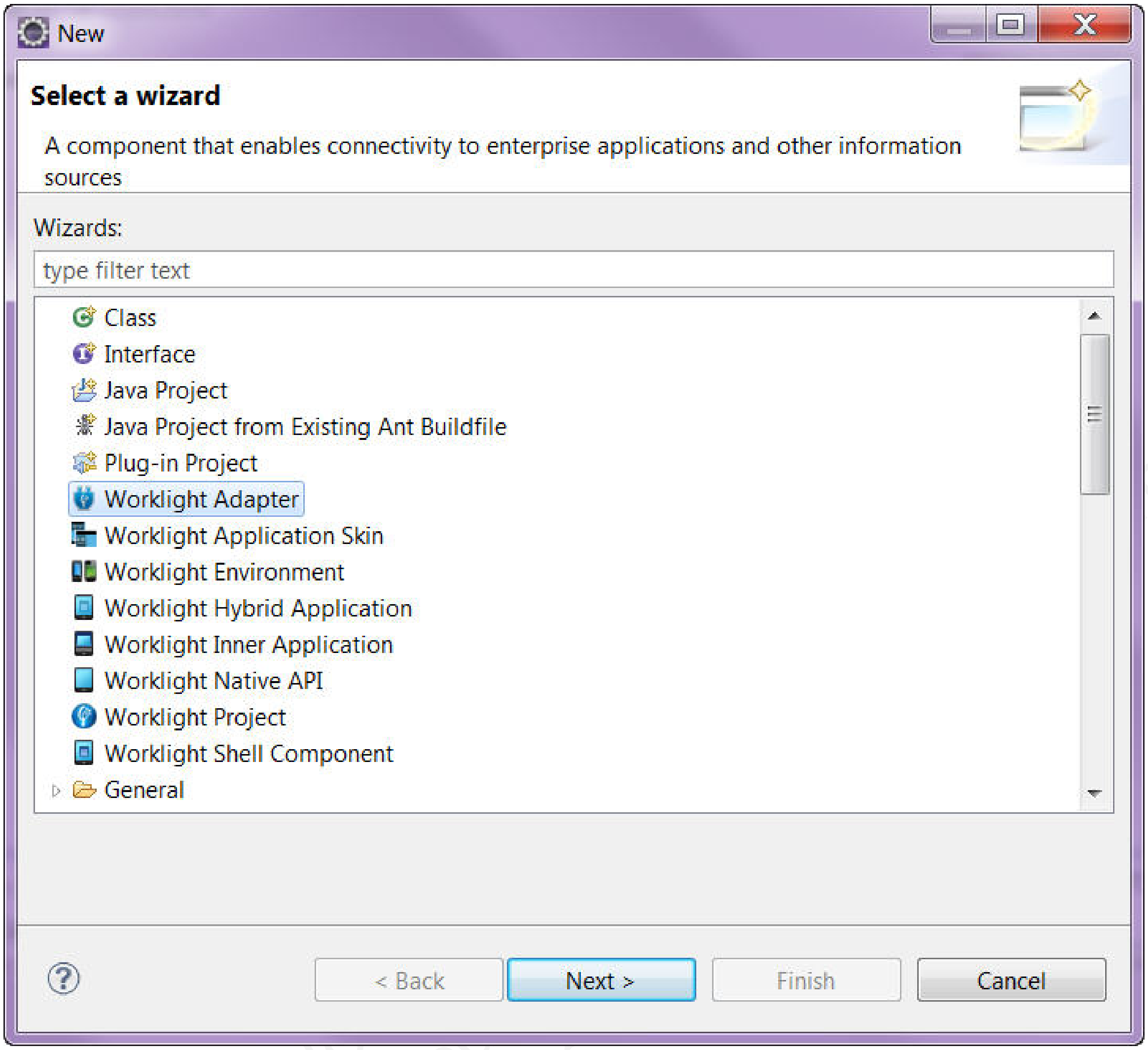
Wizards:

type filter text

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- Worklight Adapter
- Worklight Application Skin
- Worklight Environment
- Worklight Hybrid Application
- Worklight Inner Application
- Worklight Native API
- Worklight Project
- Worklight Shell Component
- General

Figure 65. 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**.



Select a wizard

A component that enables connectivity to enterprise applications and other information sources

Wizards:

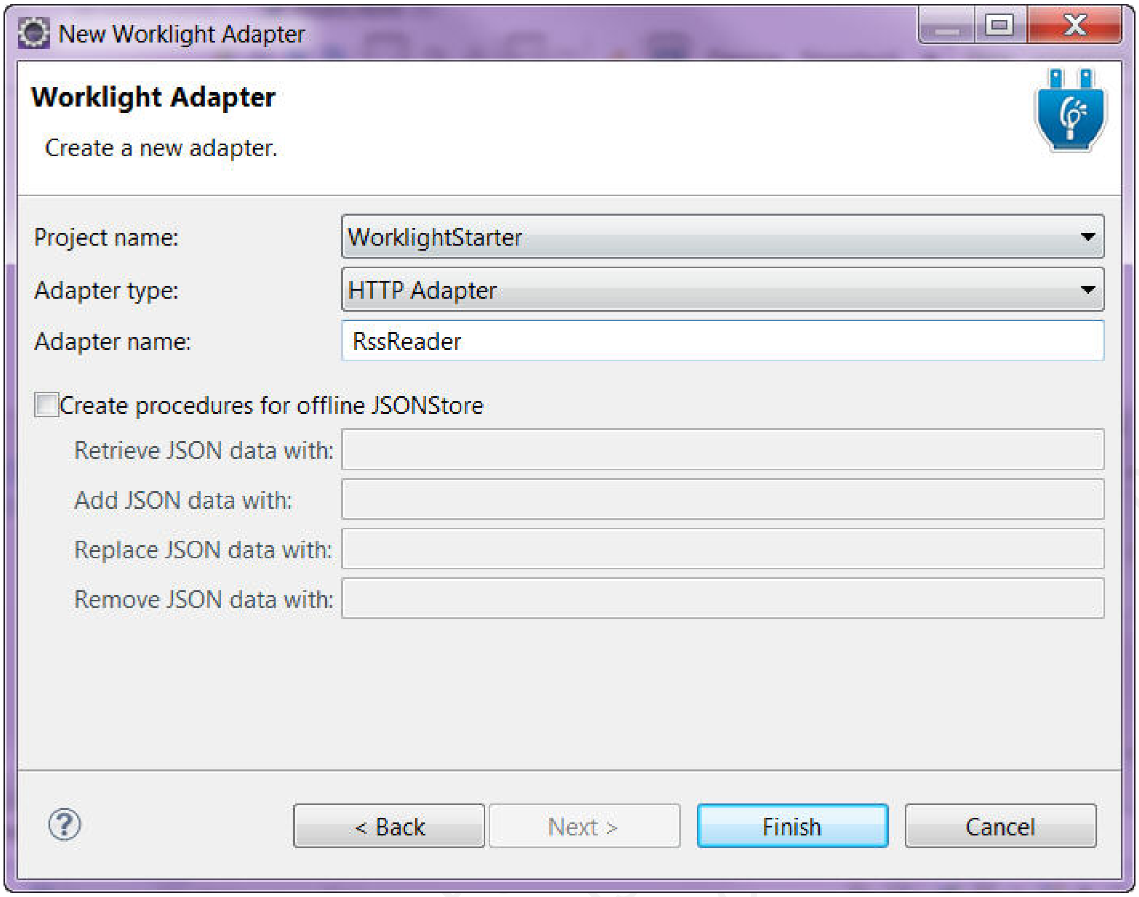
type filter text

- Class
- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- Worklight Adapter
- Worklight Application Skin
- Worklight Environment
- Worklight Hybrid Application
- Worklight Inner Application
- Worklight Native API
- Worklight Project
- Worklight Shell Component
- General

Figure 66. 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**.



New Worklight Adapter

Worklight Adapter

Create a new adapter.

Project name: WorklightStarter

Adapter type: HTTP Adapter

Adapter name: RssReader

☐ Create procedures for offline JSONStore

Retrieve JSON data with:

Add JSON data with:

Replace JSON data with:

Remove JSON data with:

Figure 67. Selecting an adapter type.

Generating adapters with the services discovery wizard

With the services discovery wizard, you specify the back-end services that you want to invoke from your IBM Worklight project, and generate the adapters that connect to those services.

About this task

The services discovery wizard supports the following types of back-end services:

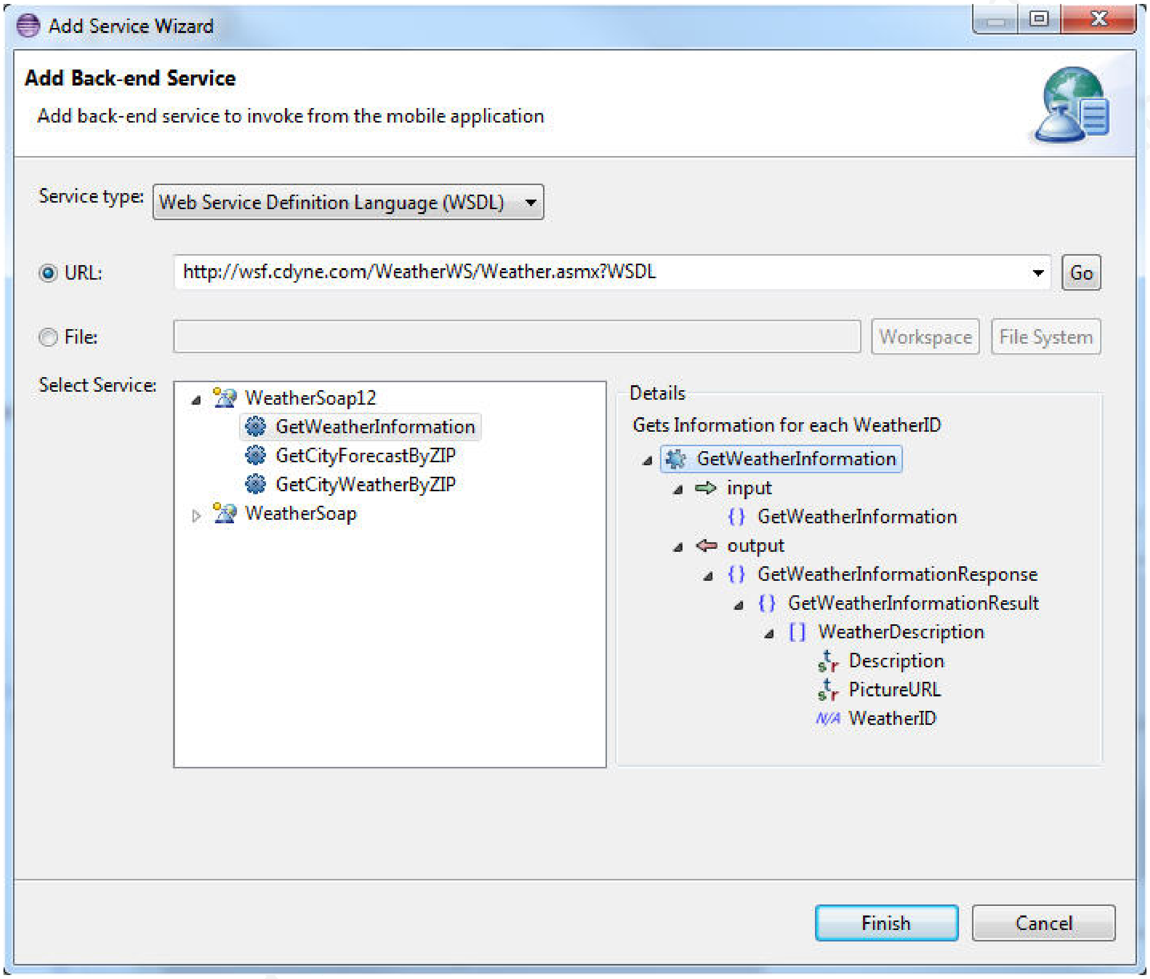
- Web Services, as described by Web Services Description Language (WSDL) files. These services are procedural in nature, with inputs and outputs that are explicit. For example, when a web service calls a remote procedure, it gets a result.
- Services that are exposed by an SAP Netweaver Gateway. These services are resource-based, which means that they expose a collection of resources that you can manipulate. Like web services, they can also have custom procedural operations, and generate inputs and outputs.

The adapters that communicate with the chosen back-end service are automatically generated, and placed in the **adapters** folder of your project.

Note: If you manually modify an adapter file, first create a copy of this file, and make sure to modify only the copied file. The services discovery wizard might regenerate the original file each time you add a service. The exact adapter that is regenerated depends on the type of service that is involved.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services** to start the services discovery wizard.
2. Select the type of service that you want to invoke from your application.
3. Depending on the selected type, define the service you want to use, as described in the following sections:
   - **WSDL** service type:



Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: Web Service Definition Language (WSDL)

● URL: http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL

○ File:

Figure 68. Adding a web service

a. Enter a URL or select one from the **URL** drop-down list, and click **Go**; or browse to a file in your workspace or in your system.

   Note: If you enter a secure URL (https), the system fetches the certificate from the specified server, and stores it into a private key storage area that is created in your workspace.

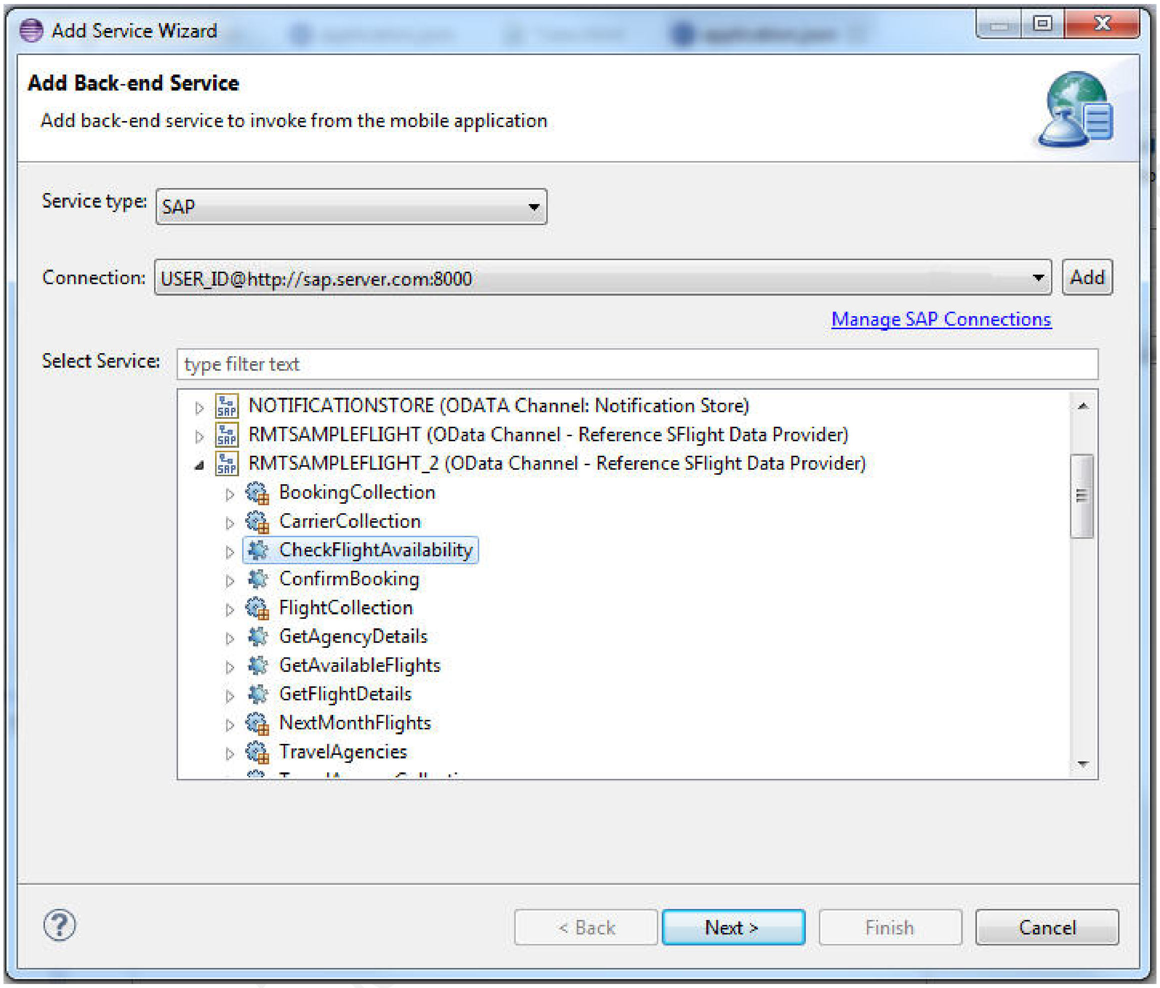
b. Optional. If you are prompted to, enter your credentials.

   You can now see the list of available services. Different types of information are displayed in the **Details** pane, depending on the level you select:

- The first level corresponds to the binding configuration details. When this level is selected, the **Details** pane shows the SOAP version.
- The second level corresponds to the data operation details. When this level is selected, the **Details** pane shows the input and the output of the remote invoked procedure.

c. Select the service that you want invoke from your application.

- **SAP Netweaver Gateway Services** service type:



Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: SAP

Connection: USER_ID@http://sap.server.com:8000

Select Service: type filter text

- NOTIFICATIONSTORE (ODATA Channel: Notification Store)
- RMTSAMPLEFLIGHT (OData Channel - Reference SFlight Data Provider)
- RMTSAMPLEFLIGHT_2 (OData Channel - Reference SFlight Data Provider)
  - BookingCollection
  - CarrierCollection
  - CheckFlightAvailability
  - ConfirmBooking
  - FlightCollection
  - GetAgencyDetails
  - GetAvailableFlights
  - GetFlightDetails
  - NextMonthFlights
  - TravelAgencies

Figure 69. Adding a service exposed by SAP

a. Set up a connection to an SAP Netweaver Gateway server by either:
   - Clicking **Add** to create an SAP connection.
   - Clicking the **Manage SAP Connections** link to edit existing connections.
   - Selecting an existing SAP connection from the **Connection** drop-down list.

b. Proceed with the connection configuration by entering your server URL, client ID, user name, and password.

   In the **Select Service** pane, you can now see the list of SAP services that are available on the server you specified.

4. Click **Finish**.

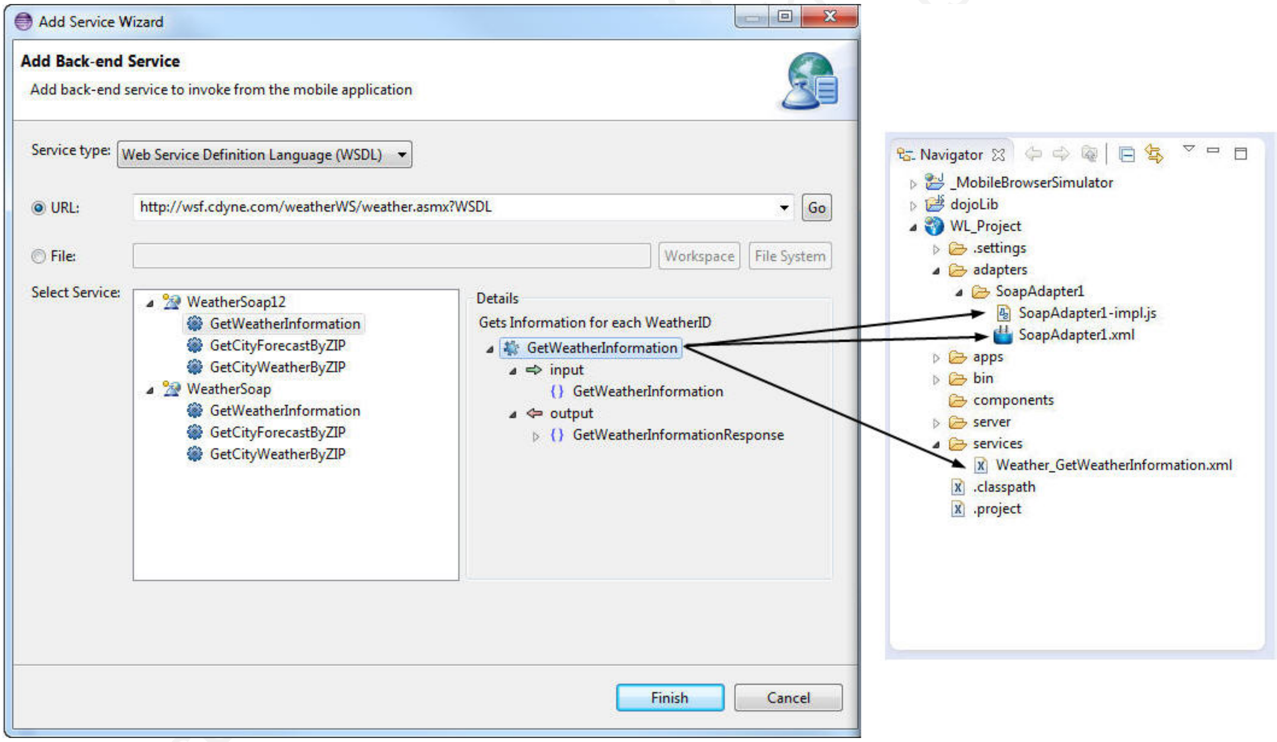
Results

An adapter is generated under the **adapters** folder of your project. You can use this adapter to invoke services with JavaScript calls.

An `.xml` service description file is also generated under the **services** folder of your project. This `.xml` file is used by IBM Worklight Application Framework (Beta code) to generate data object definitions and operation mappings (for more information, see "Developing hybrid applications with IBM Worklight Application Framework" on page 425). If you do not use IBM Worklight Application Framework, you can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

For more information about:

- invoking the generated SOAP adapters procedure, see "Invocation of generated SOAP adapters."
- the content of generated SAP adapters, see "The root element of the SAP Netweaver Gateway adapter XML file" on page 542 and "The <connectionPolicy> element of the SAP Netweaver Gateway adapter" on page 542.



Add Service Wizard

Add Back-end Service

Add back-end service to invoke from the mobile application

Service type: Web Service Definition Language (WSDL)

● URL: http://wsf.cdyne.com/weatherWS/weather.asmx?WSDL

○ File:

Figure 70. Files generated from the services discovery

Invocation of generated SOAP adapters

The generated SOAP adapters have a procedure that calls the back-end service operation. You can invoke this procedure from your IBM Worklight application in the same way as you invoke other IBM Worklight adapter procedures, by providing the necessary parameters for the invocation.

The generated procedure accepts two parameters: the message to the service, and custom HTTP headers.

The message to send to the service (required)

This mandatory parameter is the message to send to the service in JSON format.

The message parameter is a JSON representation of the XML message to include in the SOAP body that is sent to the service.

The following examples show JSON representations for sample XML messages.

1. Simple XML message: the adapter converts the provided JSON parameter into XML body by creating a matching element for each JSON attribute.

   The following JSON parameter in the procedure:

   ```
   {"GetTechnicianVisits": {"TechnicianId": "1"}}
   ```

   is transformed by the adapter into the following XML fragment in the SOAP body:

   ```
   <GetTechnicianVisits>
     <TechnicianId>1</TechnicianId>
   <GetTechnicianVisits>
   ```
2. XML messages with namespaces

   The generated adapter implementation (`SoapAdapterX-impl.js`) has a set of namespace prefixes imported from the provided WSDL service. To specify elements with specific namespaces, those prefixes must be used to name the relevant JSON attributes.

   The following JSON parameter in the procedure:

   ```
   {"tns1:GetTechnicianVisits": {"tns1:TechnicianId": "1"}}
   ```

   is transformed by the adapter into the following XML fragment in the SOAP body:

   ```
   <GetTechnicianVisits xmlns:tns1="http://namespace/sample">
     <TechnicianId>1</TechnicianId>
   </GetTechnicianVisits>
   ```
3. XML messages with attributes

   Adding the @ prefix to a JSON attribute name instructs the adapter to create an attribute instead of creating an element.

   The following JSON parameter in the procedure:

   ```
   {"GetTechnicianVisits": {"@technicianId": "1"}}
   ```

   is transformed by the adapter into the following XML fragment in the SOAP body:

   ```
   <GetTechnicianVisits technicianId="1"/>
   ```

A JSON object that holds custom HTTP headers for the invocation (optional)

This optional parameter is a JSON object that lists custom HTTP headers (key values). These custom HTTP headers are added to the service call when the POST request is invoked with the generated SOAP message.

```
{ 'custom-header-1': 'value1', 'custom-header-2': 'value2' }
```

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 94. Parameters for adapter invocation

Property	Description
`adapter`	The name of the adapter
`procedure`	The name of the procedure
`parameters`	An array of parameter values

The request can be either `GET` or `POST`.

Note: The invocation service uses the same authentication framework as described in the “IBM Worklight security framework” on page 600 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 608 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()
```

Note: Global variables are handled according to the following rules:

- In the same user session (for example, an application loaded in a browser), the values of global variables persist from one method call of an adapter to another method call of the same adapter (that is, they are not reset).
- If you create two different user sessions that connect to the same adapter (for example, by opening the same app in different browsers or devices), every user session holds its own global variable state.
- If a user session expires, the Rhino session expires, and variables are no longer defined.

Encoding a SOAP XML envelope

Follow these instructions to encode a SOAP XML envelope within a request body

About this task

When you need to invoke a SOAP-based service in an HTTP adapter, encode the SOAP XML envelope within the request body.

Procedure

Encode XML within JavaScript by using E4X E4X is officially part of JavaScript 1.6. This technology can be used to encode any XML document, not necessarily SOAP envelopes. For more information about E4X, see the related link.

Example

```
var request =
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
<S:Body>
<requestMessageObject xmlns="http://acme.com/ws/">
<messageHeader>
<version>1.0</version>
<originatingDevice>{originatingDevice}</originatingDevice>
<originatingIP>
{WL.Server.configuration["local.IPAddress"]}
</originatingIP>
<requestTimestamp>
{new Date().toLocaleString()}
</requestTimestamp>
</messageHeader>
```

```
<messageData>
<context>
<userkey>{userKey}</userkey>
<sessionid>{sessionid}</sessionid>
</context>
</messageData>
</requestMessageObject>
</S:Body>
</S:Envelope>;
```

You can use the WL.Server.signSoapMessage() method only inside a procedure declared within an HTTP adapter. It signs a fragment of the specified envelope with ID `wsId`, using the key in the specified **`keystoreAlias`**, inserting the digital signature into the input document.

To use WL.Server.signSoapMessage() API when running IBM Worklight on IBM WebSphere Application Server you might need to add a JVM argument that instructs WebSphere to use a specific **`SOAPMessageFactory`** implementation instead of a default one. To do this, you must go to **Application servers** > ***{server_name}*** > **Process definition** > **Java Virtual Machine** and provide the following argument under **Generic JVM arguments**, typing in the code phrase exactly as it is presented here:

```
-Djavax.xml.soap.MessageFactory=com.sun.xml.internal.messaging.saaj.soap.ver1_1.SOAPMessage
```

You must then restart the JVM.

Important: This workaround is only for IBM WebSphere.

Related information:

http://www.w3schools.com/xml/xml_e4x.asp

Calling Java code from a JavaScript adapter

Follow these instructions to instantiate Java objects and call their methods from JavaScript code in your adapter.

Before you begin

Attention: The name of any Java package to which you refer from within an adapter must start with the domains `com`, `org`, or `net`.

Procedure

1. Instantiate a Java object by using the `new` keyword and apply the method on the newly instantiated object.
2. Optional: Assign a JavaScript variable to be used as a reference to the newly instantiated object.
3. Include the Java classes that are called from the JavaScript adapter in your IBM Worklight project under *`Worklight Project Folder`*`/server/java`. The 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

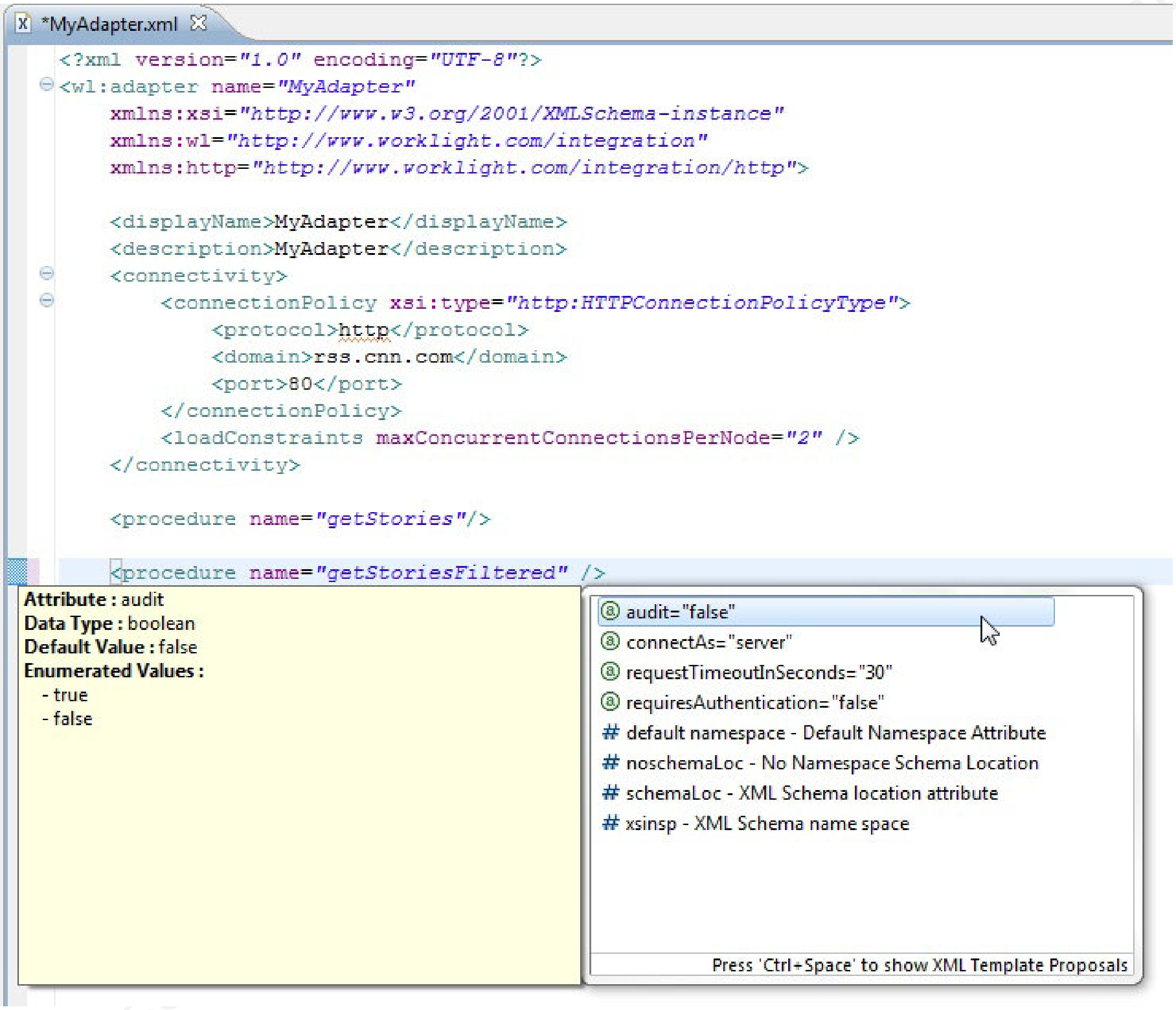
```
var x = new MyJavaClass();
var y = x.myMethod(1, "a");
```

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 following example, 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.



```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="MyAdapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>MyAdapter</displayName>
    <description>MyAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>

    <procedure name="getStoriesFiltered" />
```

Attribute : audit
Data Type : boolean
Default Value : false
Enumerated Values :
- true
- false

audit="false"
connectAs="server"
requestTimeoutInSeconds="30"
requiresAuthentication="false"
# default namespace - Default Namespace Attribute
# noschemaLoc - No Namespace Schema Location
# schemaLoc - XML Schema location attribute
# xsinsp - XML Schema name space

Figure 71. 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.

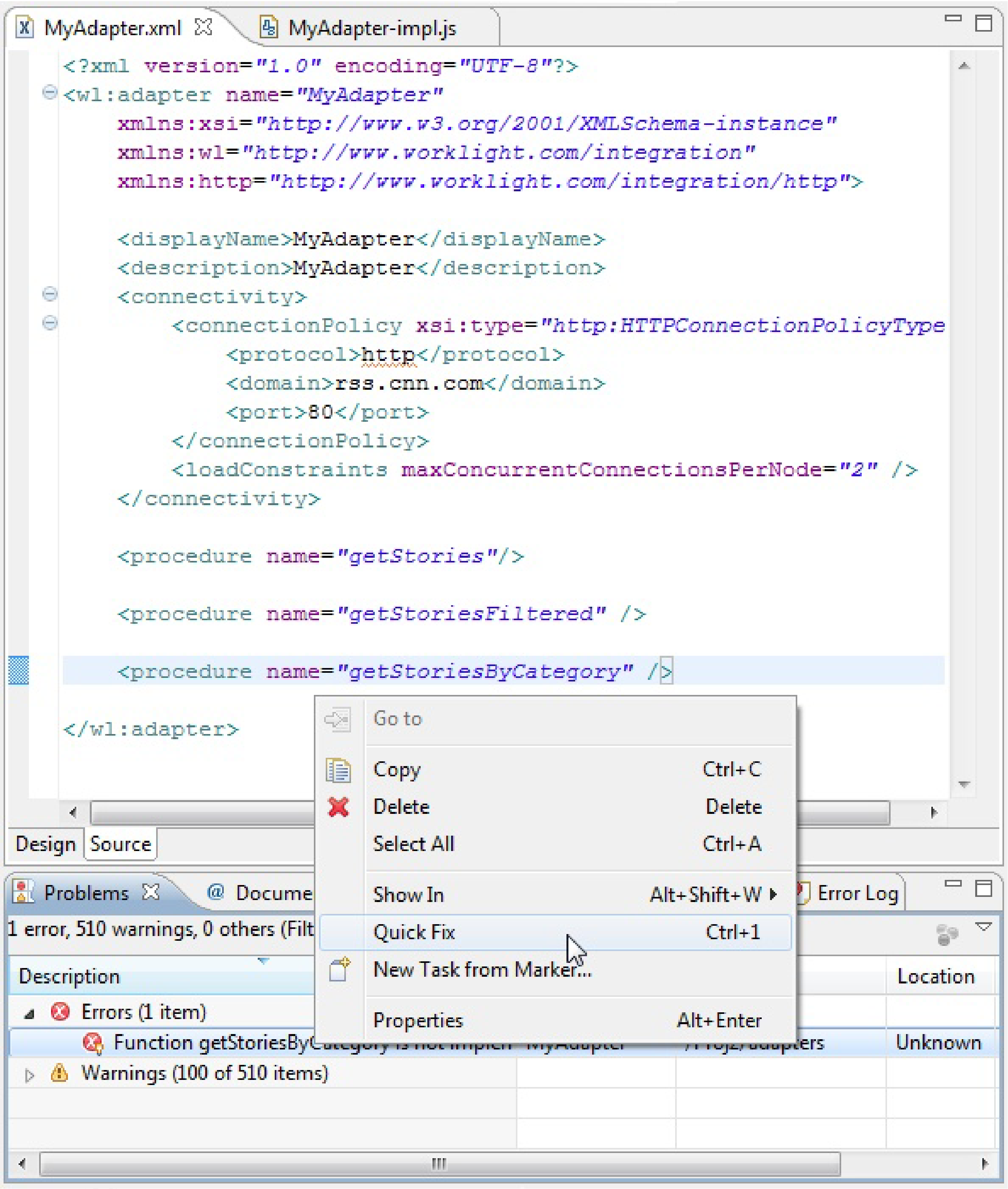
XML/JavaScript validation

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.



```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="MyAdapter"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.worklight.com/integration"
    xmlns:http="http://www.worklight.com/integration/http">

    <displayName>MyAdapter</displayName>
    <description>MyAdapter</description>
    <connectivity>
        <connectionPolicy xsi:type="http:HTTPConnectionPolicyType
            <protocol>http</protocol>
            <domain>rss.cnn.com</domain>
            <port>80</port>
        </connectionPolicy>
        <loadConstraints maxConcurrentConnectionsPerNode="2" />
    </connectivity>

    <procedure name="getStories"/>

    <procedure name="getStoriesFiltered" />

    <procedure name="getStoriesByCategory" />

</wl:adapter>
```

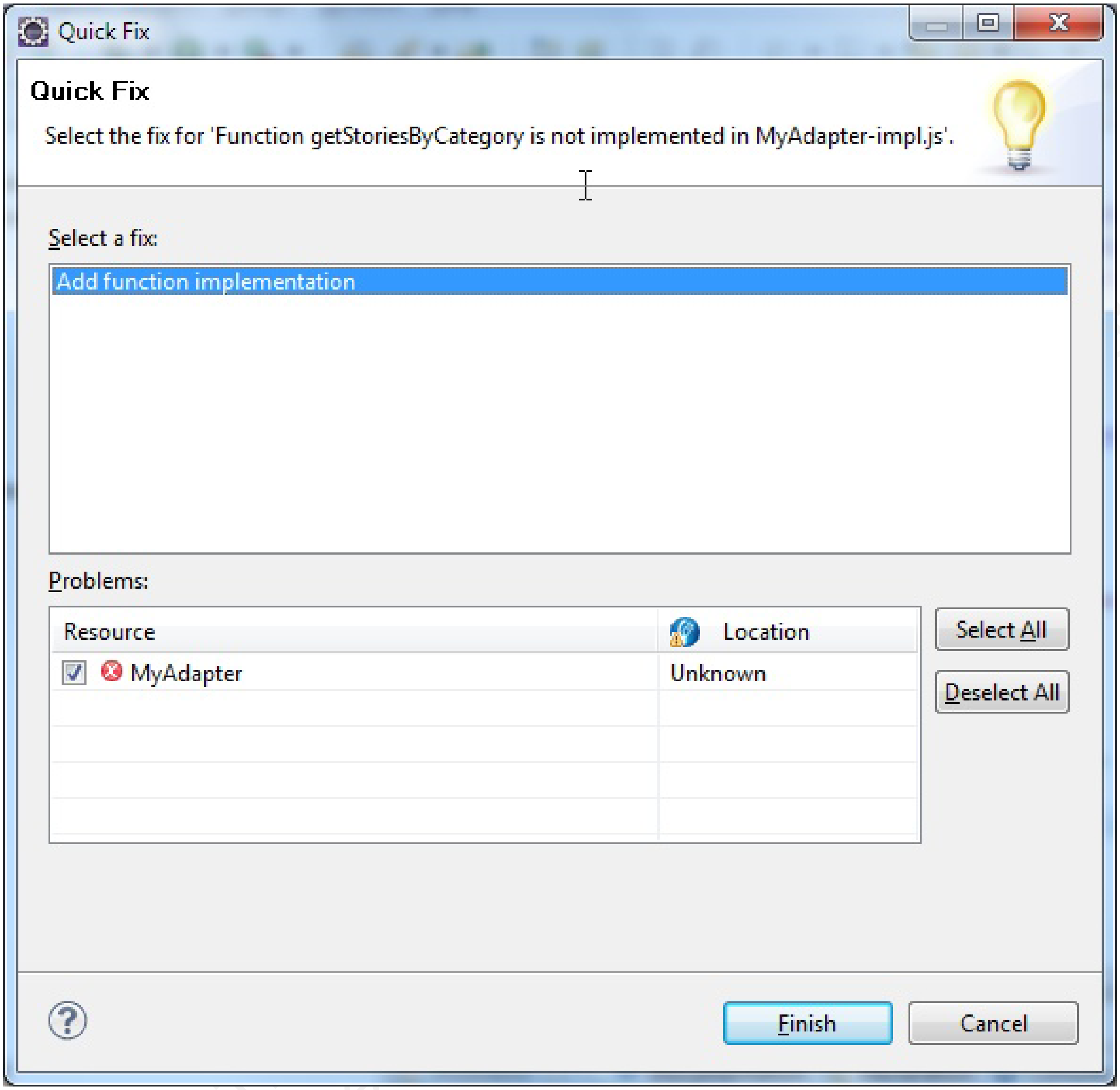
1 error, 510 warnings, 0 others (Filt

Errors (1 item)

Function getStoriesByC

Warnings (100 of 510 items)

Figure 72. Quick Fix options for adapter configuration problems.


Quick Fix

Select the fix for 'Function getStoriesByCategory is not implemented in MyAdapter-impl.js'.

Select a fix:

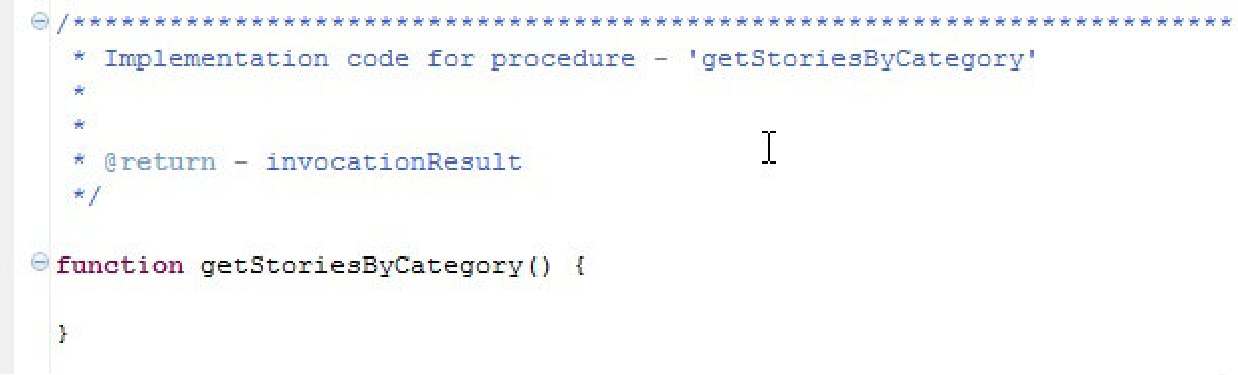
Add function implementation

Problems:

Resource	Location
☑ MyAdapter	Unknown

Figure 73. 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.

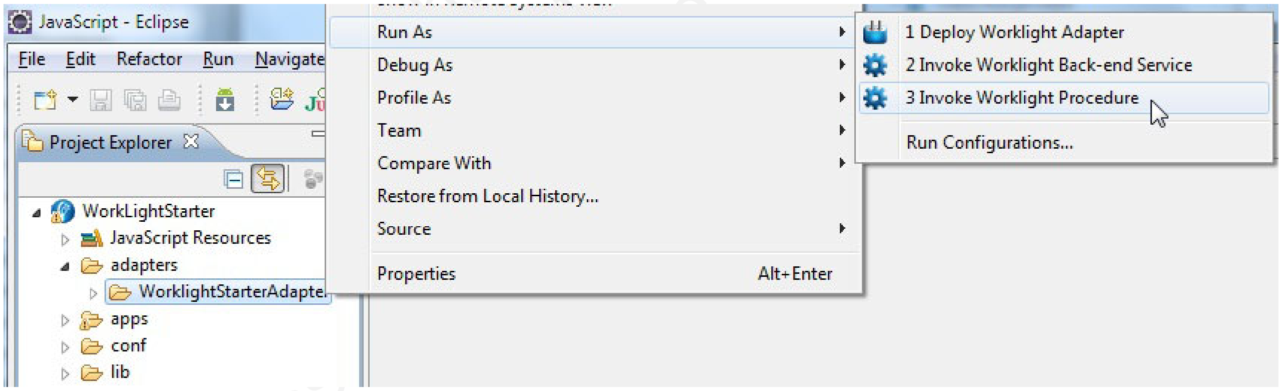
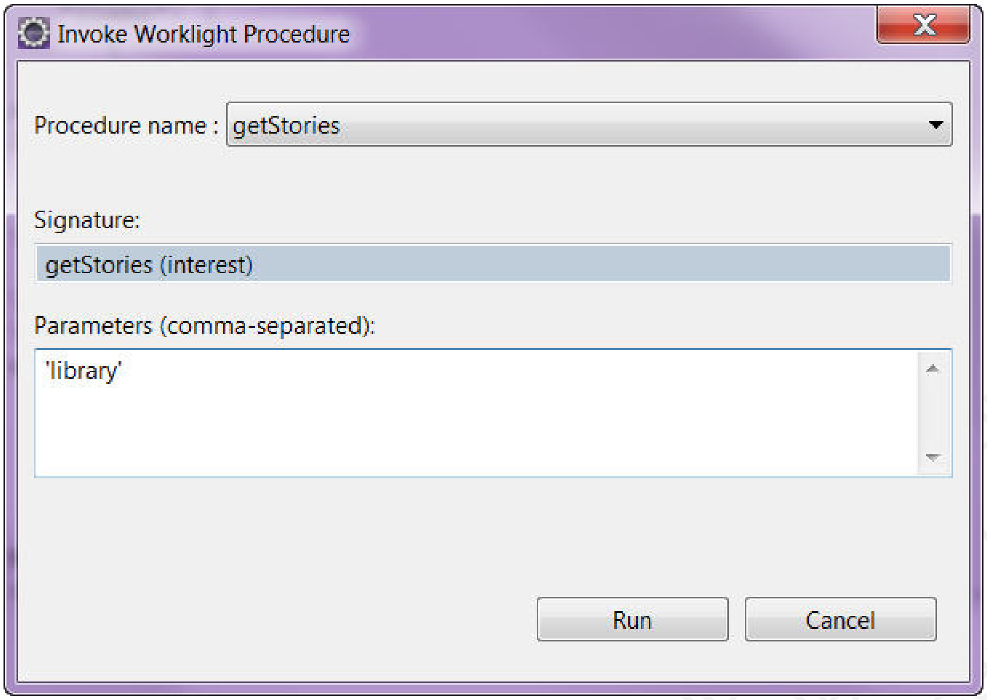


Figure 74. Invoking IBM Worklight procedures.

In the dialog box, provide a comma-separated list of procedure parameters.


Invoke Worklight Procedure

Procedure name : getStories

Signature:

getStories (interest)

Parameters (comma-separated):

'library'

Figure 75. 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**.

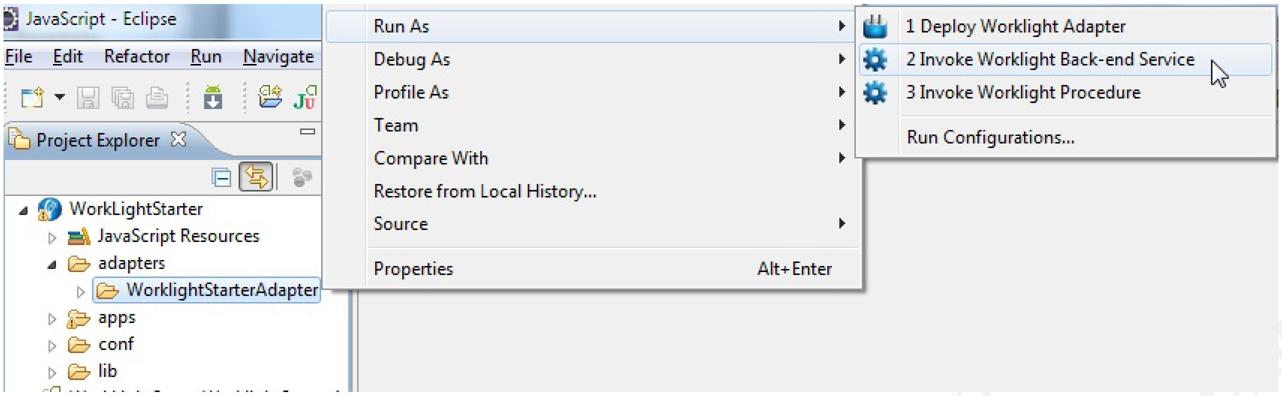
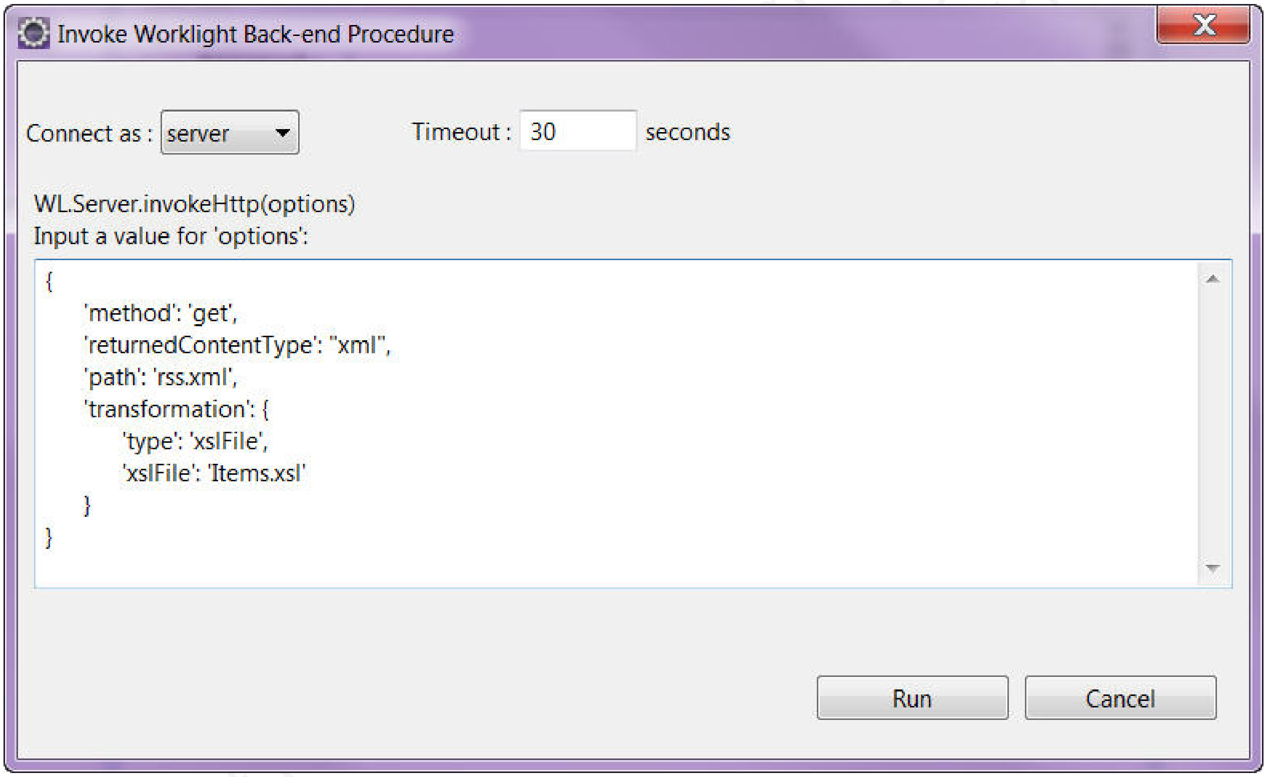


Figure 76. Invoking an IBM Worklight back-end service

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.
   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.



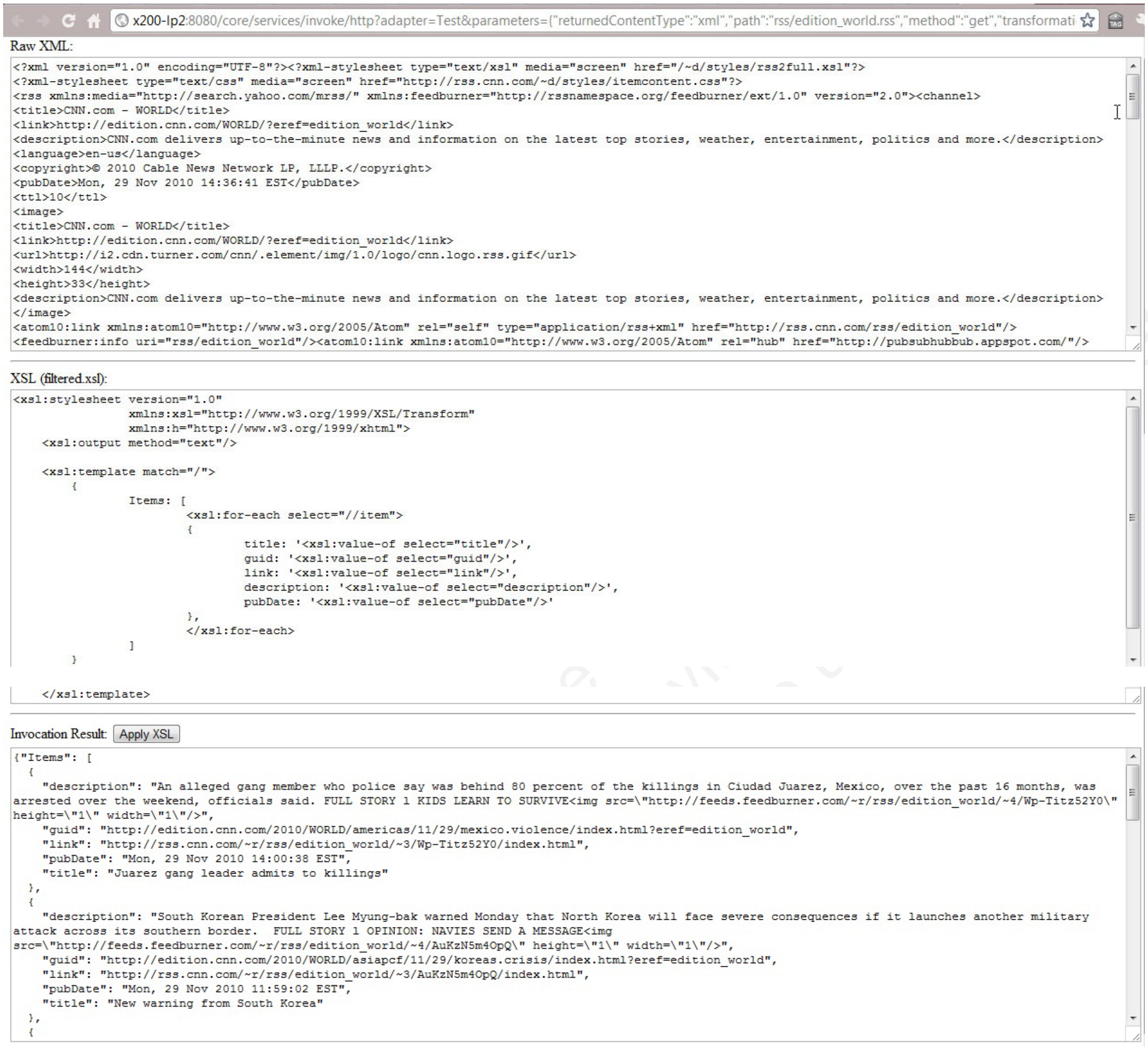
Connect as : server Timeout : 30 seconds

WL.Server.invokeHttp(options)
Input a value for 'options':

```
{
    'method': 'get',
    'returnedContentType': "xml",
    'path': 'rss.xml',
    'transformation': {
        'type': 'xslFile',
        'xslFile': 'Items.xsl'
    }
}
```

Figure 77. Invocation parameters.

3. Optional: Change the XSL transformation by editing it in the edit box, then click **Apply XSL** to regenerate the JSON format.



x200-lp2:8080/core/services/invoke/http?adapter=Test&parameters={"returnedContentType":"xml","path":"rss/edition_world.rss","method":"get","transformati

Raw XML:

```
<?xml version="1.0" encoding="UTF-8"?><?xml-stylesheet type="text/xsl" media="screen" href="/~d/styles/rss2full.xsl"?>
<?xml-stylesheet type="text/css" media="screen" href="http://rss.cnn.com/~d/styles/itemcontent.css"?>
<rss xmlns:media="http://search.yahoo.com/mrss/" xmlns:feedburner="http://rssnamespace.org/feedburner/ext/1.0" version="2.0"><channel>
<title>CNN.com - WORLD</title>
<link>http://edition.cnn.com/WORLD/?eref=edition_world</link>
<description>CNN.com delivers up-to-the-minute news and information on the latest top stories, weather, entertainment, politics and more.</description>
<language>en-us</language>
<copyright>© 2010 Cable News Network LP, LLLP.</copyright>
<pubDate>Mon, 29 Nov 2010 14:36:41 EST</pubDate>
<ttl>10</ttl>
<image>
<title>CNN.com - WORLD</title>
<link>http://edition.cnn.com/WORLD/?eref=edition_world</link>
<url>http://i2.cdn.turner.com/cnn/.element/img/1.0/logo/cnn.logo.rss.gif</url>
<width>144</width>
<height>33</height>
<description>CNN.com delivers up-to-the-minute news and information on the latest top stories, weather, entertainment, politics and more.</description>
</image>
<atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="self" type="application/rss+xml" href="http://rss.cnn.com/rss/edition_world"/>
<feedburner:info uri="rss/edition_world"/><atom10:link xmlns:atom10="http://www.w3.org/2005/Atom" rel="hub" href="http://pubsubhubbub.appspot.com/"/>
```

XSL (filtered.xsl):

```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:h="http://www.w3.org/1999/xhtml">
    <xsl:output method="text"/>

    <xsl:template match="/">
        {
                Items: [
                        <xsl:for-each select="//item">
                        {
                                title: '<xsl:value-of select="title"/>',
                                guid: '<xsl:value-of select="guid"/>',
                                link: '<xsl:value-of select="link"/>',
                                description: '<xsl:value-of select="description"/>',
                                pubDate: '<xsl:value-of select="pubDate"/>'
                        },
                        </xsl:for-each>
                ]
        }

    </xsl:template>
```

Invocation Result:

```
{"Items": [
  {
    "description": "An alleged gang member who police say was behind 80 percent of the killings in Ciudad Juarez, Mexico, over the past 16 months, was
arrested over the weekend, officials said. FULL STORY l KIDS LEARN TO SURVIVE<img src=\"http://feeds.feedburner.com/~r/rss/edition_world/~4/Wp-Titz52Y0\"
height=\"1\" width=\"1\"/>",
    "guid": "http://edition.cnn.com/2010/WORLD/americas/11/29/mexico.violence/index.html?eref=edition_world",
    "link": "http://rss.cnn.com/~r/rss/edition_world/~3/Wp-Titz52Y0/index.html",
    "pubDate": "Mon, 29 Nov 2010 14:00:38 EST",
    "title": "Juarez gang leader admits to killings"
  },
  {
    "description": "South Korean President Lee Myung-bak warned Monday that North Korea will face severe consequences if it launches another military
attack across its southern border.  FULL STORY l OPINION: NAVIES SEND A MESSAGE<img
src=\"http://feeds.feedburner.com/~r/rss/edition_world/~4/AuKzN5m4OpQ\" height=\"1\" width=\"1\"/>",
    "guid": "http://edition.cnn.com/2010/WORLD/asiapcf/11/29/koreas.crisis/index.html?eref=edition_world",
    "link": "http://rss.cnn.com/~r/rss/edition_world/~3/AuKzN5m4OpQ/index.html",
    "pubDate": "Mon, 29 Nov 2010 11:59:02 EST",
    "title": "New warning from South Korea"
  },
  {
```

Figure 78. 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 Worklight Adapter**.

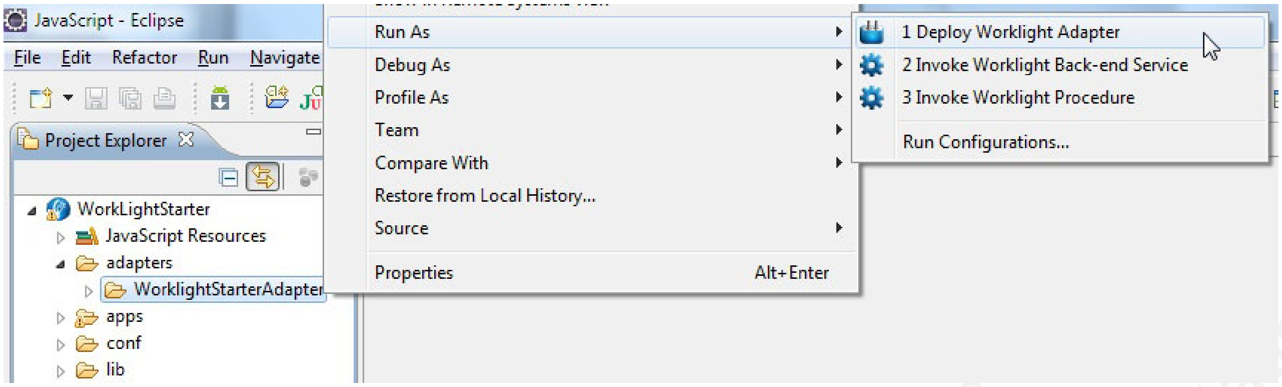


Figure 79. Deploying a Worklight adapter.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

JMS adapters

Java messaging service (JMS) is the standard messaging Java API for sending messages between two or more clients. The Worklight JMS adapter allows reading and writing to messaging providers that implement the JMS API.

The following series of topics contain instructions for configuring a JMS adapter to work with different messaging providers.

Connecting JMS adapters to a Liberty profile server

Follow these instructions to develop and test Worklight adapters that use Java Message Service (JMS) on a WebSphere Application Server Liberty Profile ND server.

Before you begin

If you want to create adapters that use the JMS API, you must understand that the WebSphere Application Server Liberty Profile included with Worklight Studio does not contain the built-in Liberty JMS features. Therefore, an embedded Worklight Development Server or a local external instance of this bundled Liberty profile server cannot act as a JMS provider.

About this task

JMS is supported by the WebSphere Application Server Liberty Profile V8.5 ND (Network Deployment) server. If you have a local copy of this application server that is installed on the same workstation as Worklight Studio, you can use it to develop and test your JMS applications.

Because WebSphere Application Server Liberty Profile does not support remote JNDI lookups, it is not possible to make remote connections to the JMS server. The IBM Worklight adapter must be running on the same local Liberty profile server that has JMS enabled.

The following procedure shows how to connect to an external Liberty profile server that supports JMS.

Procedure

1. Enable JMS on your Liberty profile ND server by using the procedures in the IBM WebSphere user documentation at Configuring point-to-point messaging for a single Liberty profile server. Make a note of the JNDI connection factory and queue name (shown in the following code example):

   ```
   <!-- Enable features -->
   <featureManager>
     <feature>jsp-2.2</feature>
     <feature>wasJmsServer-1.0</feature>
     <feature>wasJmsClient-1.1</feature>
     <feature>jndi-1.0</feature>
   </featureManager>

   <messagingEngine id="defaultME">
     <queue
       id="libertyQ"
       forceReliability="ReliablePersistent"
       maxQueueDepth="5000">
     </queue>
   </messagingEngine>

   <jmsQueueConnectionFactory jndiName="jms/libertyQCF" connectionManagerRef="ConMgr2">
     <properties.wasJms
       nonPersistentMapping="ExpressNonPersistent"
       persistentMapping="ReliablePersistent"/>
   </jmsQueueConnectionFactory>

   <connectionManager id="ConMrg2" maxPoolSize="2"/>

   <jmsQueue jndiName="jms/libertyQue">
     <properties.wasJms
       queueName="libertyQ"
       deliveryMode="Application"
       timeToLive="500000"
       priority="1"
       readAhead="AsConnection" />
   </jmsQueue>
   ```

2. Create a Worklight JMS adapter.
3. Because the adapter runs on a JMS-enabled Liberty profile server, the naming connection section of the adapter.xml file is not necessary. It can remain commented out.
4. Enter the JNDI name for the connection factory that was created in the server.xml file.

   ```
   <connectivity>
     <connectionPolicy xsi:type="jms:JMSConnectionPolicyType">

     <!-- <namingConnection url="MY_JNDI_URL"
             initialContextFactory="providers_initial_context_factory_class_name"
             user="JNDIUserName"
             password="JNDIPassword"/> -->
       <jmsConnection
         connectionFactory="jms/libertyQCF"
         user="admin"
         password="admin"
         />
     </connectionPolicy>

     <loadConstraints maxConcurrentConnectionsPerNode="10"/>
   </connectivity>
   ```

5. In the JMS adapter implementation file, enter the JNDI name for the queue as the destination for both the read and write methods:

```
function readMessage() {
  var result = WL.Server.readSingleJMSMessage({
    destination:"jms/libertyQue",
    timeout: 60
  });
  if (!result.message) {
    WL.Logger.debug(">> JMS adapter >> readNextMessage >> no message in queue");
    return {};
  } else {
    WL.Logger.debug(">> JMS adapter >> readNextMessage >> message received ::");
    return result.message;
  };
}
```

6. Change the Worklight Target Server in Worklight Studio to point to your Liberty ND server. Documentation for this procedure can be found at “Working with multiple Worklight Servers in Worklight Studio” on page 345.
7. Build and deploy the Worklight adapter to the Liberty profile ND server. You can test the JMS adapter in your browser by using the following URL syntax:

   `http://<liberty-hostname>:<port>/<context-root>/invoke?adapterName=<adapterName>&procedure=<procedureName>&parameters=["<parameters>"]`

   An example of a URL pointing to an external Liberty profile ND server:

   `http://localhost:9080/worklight/invoke?adapter=JMSAdapter&procedure=writeMessage&parameters=["Hello World"]`

Connecting a Worklight JMS adapter to WebSphere MQ

Follow these instructions to connect an IBM Worklight Java Message Service (JMS) adapter to WebSphere MQ.

Before you begin

Ensure that you have prior knowledge of WebSphere MQ and have a WebSphere MQ Message Broker setup with the appropriate JMS administered objects. For more information about setting up WebSphere MQ for JMS, see the IBM WebSphere MQ user documentation.

About this task

The Worklight JMS adapter does not support connecting to WebSphere MQ through bindings mode, only client mode. A TCP connection is created for each JMS request, even if the JMS broker and Worklight adapter are running on the same computer.

The following procedure shows how to connect a Worklight JMS adapter to WebSphere MQ.

Procedure

Include the required WebSphere MQ Java libraries

1. Create a new Worklight project.
2. Locate the `java/lib` directory in your WebSphere MQ directory.

   Example:

   `/opt/mqm/java/lib`
3. Copy the following JAR files from the `java/lib` directory into the `server/lib` directory of your Worklight Project:
   - `CL3Export.jar`
   - `CL3Nonexport.jar`

- com.ibm.mq.axis2.jar
- com.ibm.mq.commonservices.jar
- com.ibm.mq.defaultconfig.jar
- com.ibm.mq.headers.jar
- com.ibm.mq.jar
- com.ibm.mq.jmqi.jar
- com.ibm.mq.jms.Nojndi.jar
- com.ibm.mq.pcf.jar
- com.ibm.mq.postcard.jar
- com.ibm.mq.soap.jar
- com.ibm.mq.tools.ras.jar
- com.ibm.mqjms.jar
- connector.jar
- dhbcore.jar
- fscontext.jar
- jta.jar
- providerutil.jar
- rmm.jar

Modify the adapter xml file

4. Create a Worklight JMS adapter.
5. Open the `adapter.xml` file.
6. In the `namingConnection` element of the xml file, set the URL to the location of your bindings file that was generated by WebSphere MQ.

   Example:

   `url="file:/home/user/JMS"`
7. In the `namingConnection` element of the xml file, set the `initialContextFactory` attribute to `com.sun.jndi.fscontext.RefFSContextFactory`.
8. In the `jmsConnection` element, set the `connectionFactory` attribute to the name of the connection factory that was set up in WebSphere MQ.
9. Optional: If you have security that is enabled in WebSphere MQ, include the credentials as shown in the following code example.

```
<connectivity>
  <connectionPolicy xsi:type="jms:JMSConnectionPolicyType">
    <namingConnection
      url="file:/home/user/JMS"
      initialContextFactory="com.sun.jndi.fscontext.RefFSContextFactory"
      user="admin"
      password="password"/>
    <jmsConnection
      connectionFactory="myConnFactory"
      user="admin"
      password="password"/>
  </connectionPolicy>
  <loadConstraints maxConcurrentConnectionsPerNode="10"/>
</connectivity>
```

Modify the adapter implementation file

10. Open the adapter′s implementation file.

11. In the auto-generated read and write methods, replace the `destination` property with the name that was configured in your JMS administered object in WebSphere MQ.

    Example:

    ```
    function readMessage() {
      var result = WL.Server.readSingleJMSMessage({
        destination: "JMS1",
        timeout: 60
      });
      WL.Logger.debug(result);
      if (result.errors) {
        WL.Logger.debug(">> JMS adapter >> readNextMessage >> errors occured");
        return result;
      } else if (!result.message) {
        WL.Logger.debug(">> JMS adapter >> readNextMessage >> no messages in queue");
        return result;
      } else {
        WL.Logger.debug(">> JMS adapter >> readNextMessage >> message received ::");
      };
    }
    ```

Results

The Worklight JMS adapter is now properly configured to connect to WebSphere MQ. You can test the JMS adapter in your browser by using the following URL:

```
http://<hostname>:<port>/<context-root>/invoke?adapterName=<adapterName>&procedure=
<procedureName>&parameters=['<parameters>']
```

Example

```
http://localhost:10080/worklight/invoke?adapter=JMSAdapter&procedure=
writeMessage&parameters=['Hello World']
```

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 in the API reference section.

Here is a high-level summary of what JSONStore provides:

- A developer-friendly WL.JSONStore class 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 matches 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 581. For a table summarizing the JSONStore features, see WL.JSONStore.

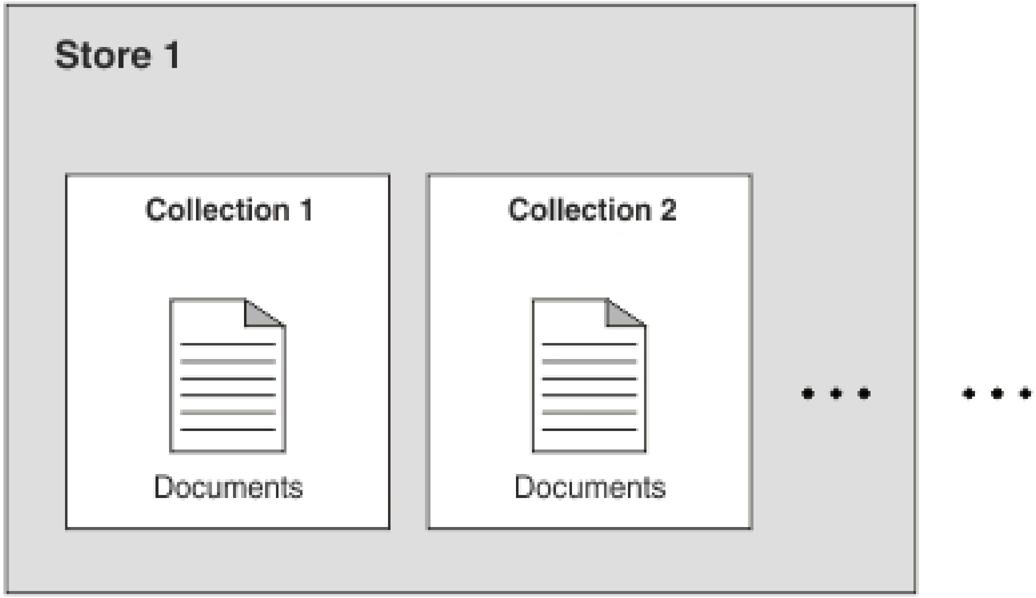


Figure 80. A basic graphic representation of JSONStore.

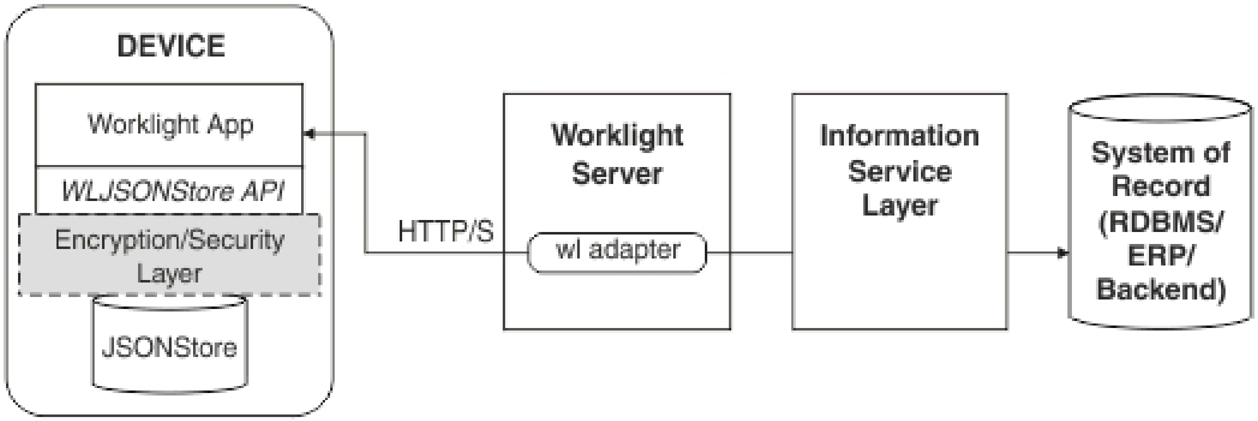

DEVICE
Worklight App
WLJSONStore API
Encryption/Security Layer
JSONStore
HTTP/S
Worklight Server
wl adapter
Information Service Layer
System of Record (RDBMS/ ERP/ Backend)

Figure 81. 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 95. A comparison of data storage technologies.

	JSONStore	**Encrypted Cache**	**LocalStorage**	**IndexedDB**	**Cordova Storage**	**Cordova File**
Android Support	⌂	⌂	⌂	⌂	⌂	⌂
iOS Support	⌂	⌂	⌂	⌂	⌂	⌂
Web	*Dev Only* (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

Since IBM Worklight V6.0, JSONStore is an optional feature. To use JSONStore, you must take steps to enable it.

About this task

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 **Run on Worklight Development Server**.

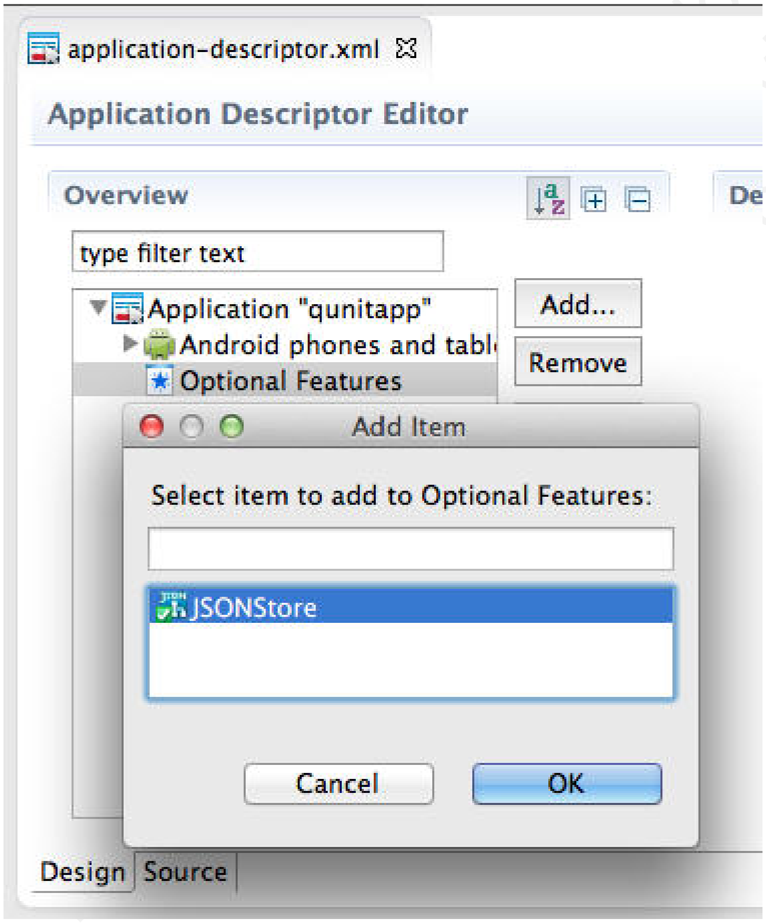


Figure 82. Enabling JSONStore

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 the `add`, `replace`, and `remove` methods in the JSONStoreInstance class 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 96. Contents of a store in JSONStore

_id	key	name	age	JSON
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. These asynchronous operations return a `jQuery promise` that is resolved when completed successfully and rejected when a failure occurs. These promises are similar to success and failure callbacks.

A `jQuery Deferred` is a promise that can be resolved or rejected. The two examples provided 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

```
var asyncOperation = function () {
  var deferred = $.Deferred();
  setTimeout(
    function () {deferred.resolve('Hello');},
    1000);
  return deferred.promise();
}
// The function passed to .then is executed after 1000 ms.
asyncOperation.then(function (response) {
  // response = 'Hello'
});
```

Using a callback instead of a promise

```
var 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 backward compatibility. It is possible but not a good practice to pass an `options` object to methods like `add`, `find`, `replace`, and `remove` in the JSONStoreInstance class, 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.

Related reference:

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

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: the find and remove methods in the JSONStoreInstance class) 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 JSONStore success and failure events and capture associated 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 the `find`, and `add` methods in the JSONStoreInstance class) 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.”

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 the `push` method in the JSONStoreInstance class) 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 the `init` method in the WL.JSONStore class 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 27.

-12 INVALID_SEARCH_FIELD_TYPES
Check that the types that you are passing to the `searchFields` are **stringinteger**,**number**, or**boolean**.

-11 OPERATION_FAILED_ON_SPECIFIC_DOCUMENT
An operation on an array of documents, for example the `replace` method in the JSONStoreInstance class, 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 the `closeAll` method in the WL.JSONStore class first. There can be only one user at a time.

-5 DESTROY_REMOVE_PERSISTENT_STORE_FAILED
A problem with the `destroy` method in the WL.JSONStore class while trying to delete the file that holds the contents of the store.

-4 DESTROY_REMOVE_KEYS_FAILED
Problem with the `destroy` method 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 the `destroy` method or the `removeCollection` method in the WL.JSONStore class before calling the `init` method 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 the `init` method.

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 the `push` and `load` methods in the JSONStoreInstance class, an adapter must be passed to the `init` method.

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 the `enhance` method in the JSONStoreInstance class to replace an existing function (`find` and `add`) 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 the `init` method 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 the `init` method 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 the `add` method.

20 UNDEFINED_PUSH_OPERATION
There is no procedure defined for pushing dirty documents to the server. For example: the `init` method (new document is dirty, operation = 'add') and the `push` method (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 `init` method.

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 the `closeAll` method.

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 the `destroy` method.

26 ERROR_CLEARING_COLLECTION
Generic Error. An error occurred in when native code called the `removeCollection` method.

27 INVALID_PARAMETER_FOR_FIND_BY_ID
Validation error.

JSONStore support

How to get support for JSONStore

- Read “JSONStore overview” on page 567 and WL.JSONStore.
- Read the Chapter 3, “Tutorials and samples,” on page 27 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 the `destroy` method in the WL.JSONStore class to start with a clean system.
- Make sure that 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 the `init` method in the WL.JSONStore class).
- 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 user name to the init method, the file is called `[username].sqlite`, for example `carlos.sqlite`. When no user name is passed, `jsonstore` is the default user name.
- (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 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 you allow 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 an appropriate 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`. They are stored as 1 and 2 when the type is `integer`.
- If an app is forced to stop or crashes, when the app is started again and `WL.JSONStore.init()` is called, it always fails with error code -1. If that happens, call the `closeAll` method in the WL.JSONStore class, followed by the `init` method.

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 includes 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 get only 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 you call the `find` method in the JSONStoreInstance class. Instead of trying to get all the documents from a local storage, get a subset and implement pagination. Using `{exact: true}` to avoid fuzzy searching and setting a limit can also help improve the `find` method's 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 that is stored inside JSONStore into various collections. Have 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 you call the `add` method in the JSONStoreInstance class with an array of objects or by using the `load` method in the JSONStoreInstance class 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 the `add` method instead of an array with 1000 documents.
- JavaScript engines have a garbage collector. Allow it to work, but do not 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 the `init` method in the WL.JSONStore class affects performance when you call the `add` method, which does the indexing. Only index the values that are used in queries for the `find` method.
- 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 tracks local changes to its documents. This behavior can be disabled, thus saving a few cycles, by passing the `{push: false}` flag to the `add`, `remove`, and `replace` methods in the JSONStoreInstance class.

Encryption and size of the collection

- Turning on encryption affects the performance of several JSONStore operations, making them take longer. For example, `init` is slower with encryption, as it must generate the encryption keys that it uses for encryption and decryption.
- The following functions are affected by both encryption and the size of the JSONStore collection:
  - the `count`, `replace`, and `remove` methods in the JSONStoreInstance class: These functions depend on the collection size as they must go through the whole collection to replace and remove all occurrences. Because it must go through each record, it must decrypt every one of them, which makes it much slower when encryption is used. This performance degradation is noticeable on large collections. For the count method, if you want the count of a huge encrypted

collection, you can keep a related variable that keeps the count for that collection. Update it every time that you store or remove things from the collection.

  - the `find` by query and `findAll` methods in the JSONStoreInstance class: These functions are affected by encryption, since they must decrypt every document to see whether it is a match or not. For find by query, if a limit is passed, it is potentially faster as it stops when it reaches the limit of results. JSONStore does not need to decrypt the rest of the documents to figure out if any other search results remain.

Miscellaneous

- Most of the core JSONStore API is asynchronous. Ideally, 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, the find method is finished.

JSONStore multiple user support

With JSONSTore, you can create multiple stores that contain different collections in a single IBM Worklight application.

The `init` method in the WL.JSONStore class 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 `closeAll` method in the WL.JSONStore class removes access to all the collections. It is also possible to change the password of an encrypted store calling the `changePassword` method in the WL.JSONStore class.

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 `init` method in the WL.JSONStore class. 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 you cannot 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 the `changePassword` method in the WL.JSONStore class.

The encrypted data protection key (DPK) is the key that is used to decrypt the contents of the store. The DPK is kept in the iOS keychain even if the application is uninstalled. To remove both the key in the keychain and everything else that JSONStore put in the app, call the `destroy` method in the WL.JSONStore class. This process is not applicable to Android because the encrypted DPK is stored in shared preferences and wiped out when the app is uninstalled.

Worklight adapter integration for JSONStore

You can enable JSONSore 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 27.

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 the `add` method, the `replace` method, and the `remove` method in the JSONStoreInstance class, and most other operations are specific to the local copy of the data. JSONStore tracks modifications to data made locally. The exceptions are the `push` method and the `load` method in the JSONStoreInstance class, 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 the `invokeProcedure` method as defined in the WL.Client class, `jQuery.ajax` to transmit and receive data, and the `getPushRequired` method in the JSONStoreInstance 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.

Nest 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 the push method and the load method in the JSONStoreInstance class, return an error.

When the load method in the JSONStoreInstance class 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 the getPushRequired method as defined in the JSONStoreInstance class 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 documented that was added locally).

The option {push: false} can be passed to the add method, the replace method, and the remove method as defined in the JSONStoreInstance class 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 isPushRequired method as defined in the JSONStoreInstance class. Checking how many documents are dirty can be achieved with the pushRequiredCount method as defined in the JSONStoreInstance or counting the documents returned by the getPushRequired method as defined in the JSONStoreInstance class.

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 582 for an overview. Using the wizard is optional. You can link a collection to an adapter by manually writing the procedure names when the `init` method in the WL.JSONStore class 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 `push` method and the `pushSelected` method as defined in the JSONStoreInstance class operate similarly, with several differences. The `push` method takes no parameters, and synchronizes all the documents that were modified locally. The `pushSelected` method as defined in the JSONStoreInstance class 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 (`add`, `replace`, or `remove` in JSONStoreInstance). 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 575.

Push notification

Push notification is the ability of a mobile device to receive messages that are pushed from a server. The most common form of notification is SMS (Short Message Service). Notifications are received regardless of whether the application is currently running.

Notifications can take several forms, and are platform-dependent:

- Alert: a pop-up text message
- Badge, Tile: a graphical representation that includes a short text or image
- Banner, Toast: a pop-up text message at the top of the device display that disappears after it has been read
- Audio alert

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. The unified push notification mechanism in IBM Worklight makes the entire process of communicating with the users and devices completely transparent to the developer.

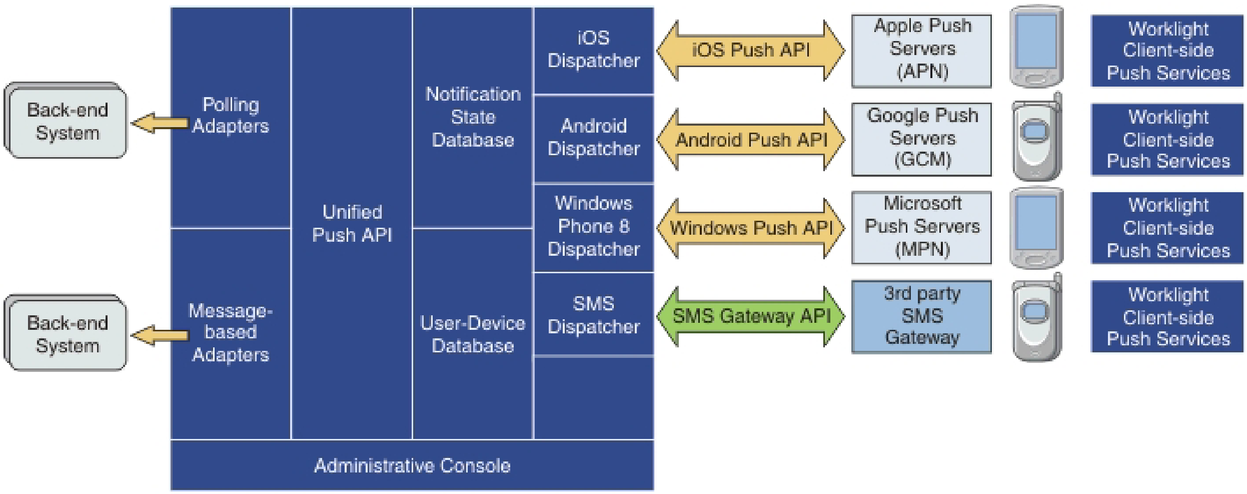


Figure 83. Push notification mechanism

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 8, Java ME, and BlackBerry devices that support SMS functions. For more information about setting up push notification for each platform, see Setting up push notifications.

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 more information, see "Configuration of IBM Worklight applications on the server" on page 772.

Architecture

Unlike other IBM Worklight services, the push server requires outbound connections to Apple, Google, and Microsoft servers using ports that are 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.

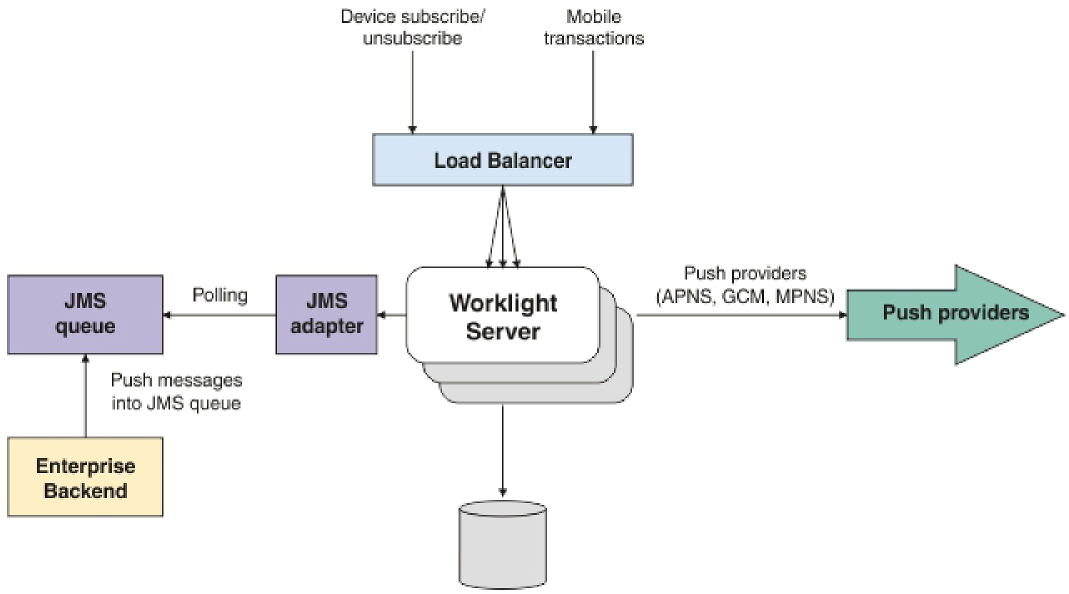


Figure 84. JMS polling push notification architecture

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 an IBM 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 an IBM Worklight adapter procedure.

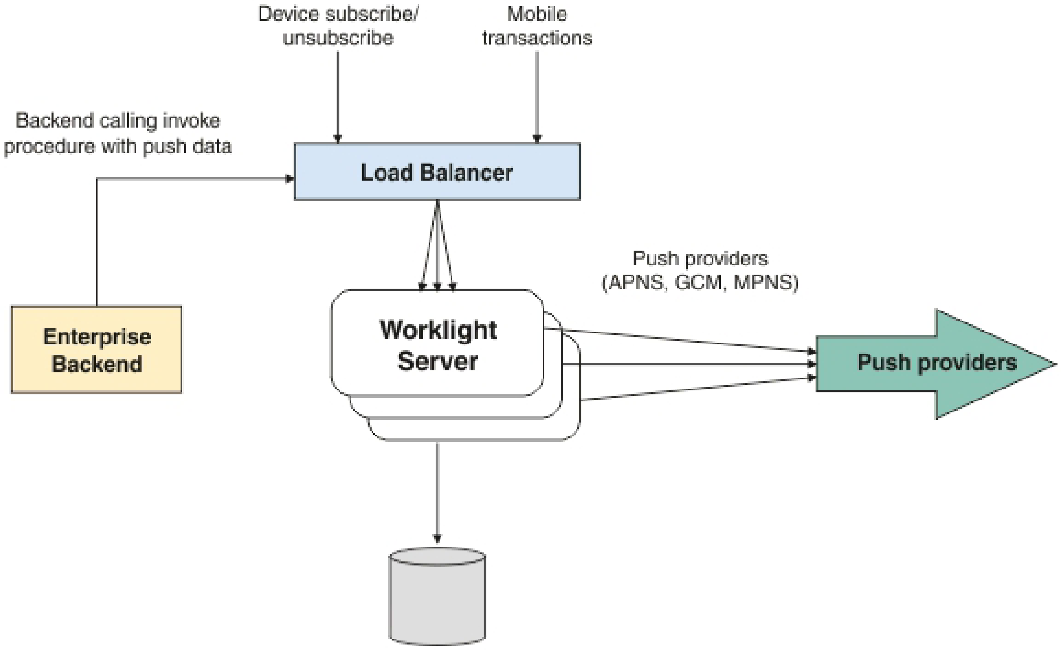

Device subscribe/ unsubscribe
Mobile transactions
Backend calling invoke procedure with push data
Load Balancer
Push providers (APNS, GCM, MPNS)
Enterprise Backend
Worklight Server
Push providers

Figure 85. 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 IBM Worklight adapter procedure that is invoked call the backend for a batch of messages rather than single messages.

Setting up push notifications

You can send push notifications to mobile devices via the Worklight Server. You can set up push notifications on Android, iOS, and Windows Phone 8.

About this task

The process for setting up push notifications varies significantly for each platform, and for Android and iOS you must refer to documentation for those products. For more information about the processes for each platform, see the following tasks:

Setting up push notifications for Android

To set up push notifications for Android devices, you must use the Google Cloud Messaging (GCM) service. In order to use GCM, you need a valid Gmail account.

Procedure

1. Create a Gmail account.
   a. Open the GCM web page: http://developer.android.com/google/gcm/gs.html.
   b. Open the Google APIs Console page. If you have not done this previously, you will be asked to create a project. The project has an ID; this is the `senderID` value that you use in the `application-descriptor.xml` file.
   c. Click the option to create a new key, and choose the option to create a browser key. The generated key is the `key` value that you use in the `application-descriptor.xml` file. When you create the browser key, do not restrict it to any specific URL.
2. From Worklight Server, ensure that ports 5228, 5229, and 5230 are open. Typically, GCM uses only port 5228, but it sometimes uses 5229 and 5230.
3. Ensure that your firewall will accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169.
4. In the `application-descriptor.xml` file, for **`<android>`** set the following attributes for the **`<pushsender>`** element:

Attribute	Description
`key`	The key value received from GCM.
`senderID`	The project ID received from GCM.

Note: Android OS 2.3.x devices must be synchronized with a Gmail account. Android OS 4.x does not impose account synchronization.

Results

Your push notifications setup is now complete.

Setting up push notifications for iOS

To set up push notifications for iOS devices, you must use the Apple Push Notification Service (APNS). In order to use APNS, you must be a registered Apple iOS Developer, and obtain an Apple APNS certificate for your application.

Before you begin

Ensure that the following servers are accessible from Worklight Server:

- Sandbox servers:
  - gateway.sandbox.push.apple.com:2195
  - feedback.sandbox.push.apple.com:2196
- Production servers:
  - gateway.push.apple.com:2195
  - feedback.push.apple.com:2196

Procedure

1. Perform the required steps to obtain your APNS certificate and its password. For detailed information, see the developerWorks article Understanding and setting up artifacts required to use iOS devices and APNS in a development environment
2. Place the Apple APNS certificate file at the root of the application folder, in which the `application-descriptor.xml` file is held.
3. Install the Entrust CA root certificate using SSL port 443. For more information, see the iOS Developer Library.
4. In the `application-descriptor.xml` file, for **`<iPhone>`** set the following attributes for the **`<pushsender>`** element:

Attribute	Description
`password`	The APNS certificate password received from Apple.

Results

Your push notifications setup is now complete.

Setting up push notifications for Windows Phone 8

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.

Before you begin

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. 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/.

Procedure

In the `application-descriptor.xml` file, for `<windowsPhone8>` set the following attributes for the `<pushSender>` element:

Attribute	Setting
`serviceName`	The common name (CN) found in the MPNS certificate's Subject value.
`keyAlias`	The alias used to access the keystore specified by the following properties in the `worklight.properties` file: • ssl.keystore.path • ssl.keystore.type • ssl.keystore.password
`keyAliasPassword`	The password for your key alias.

Results

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 byWorklight 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.

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, and search for "push notification service response codes".

Example

```
<windowsPhone8>
  <pushSender>
    <authenticatedPush serviceName="myservice"
                       keyAlias="janedoe"
                       keyAliasPassword="a1b2c3d4"</authenticatedPush>
  </pushSender>
...
</windowsPhone8>
```

Subscribing to push notifications

Before a device can start receiving push notifications, it must first subscribe to a push notification event source. When the user approves the push notification subscription, the device is registered with an appropriate push server.

About this task

There are two levels of subscription: user subscription and device subscription.

- *User subscription* is an entity that contains a user ID, a device ID, and an event source ID. It represents the intent of the user to receive notification from a specific event source.
- A *device subscription* belongs to a user subscription, and exists in the scope of a specific user and event source. A user subscription can have several device subscriptions.

The user subscription for an event source is created when the user first subscribes to the event source from any device. The event source is declared in the IBM Worklight adapter that is used by the application for push notification services.

After the user approves a push notification subscription, the device is registered with an Apple, Google, or Microsoft push server to obtain a token that is used to identify the device. The token is in the following form: `Allow notifications for application X on device Y`. The device then sends a subscription request to the Worklight Server.

Procedure

1. When a device calls the **WL.Client.Push.subscribe** API, the device registers with a push service mediator and obtains a device token. This process is done automatically by IBM Worklight.
2. When the token is obtained, the application subscribes to an event source.

3. Optional: If push notifications are no longer required, you can unsubscribe. The device subscription is deleted either by an application that calls the **WL.Client.Push.unsubscribe** API, or when the push mediator informs the Worklight Server that the device is permanently not accessible.

Results

While the user subscription exists, the Worklight Server can produce push notifications for the subscribed user. These notifications can be delivered by the adapter code to all or some of the devices that the user has subscribed from.

Web-based SMS subscription

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. For more information about how to create an event source, see the `createEventSource` method in the WL.Server class.

Table 97. Subscribe SMS servlet URL parameters

URL parameter	URL parameter description
option	Optional string. Subscribe or unsubscribe action to perform. The default option is `subscribe`. If any non-blank string other than `subscribe` is supplied, the unsubscribe action is performed.
eventSource	Mandatory string. The name of the event source. The event source name is in the format *AdapterName.EventSourceName*.
alias	Optional string. A short ID defining the event source during subscription. This ID is the same ID as provided in WL.Client.Push.subscribeSMS.
phoneNumber	Mandatory string. User phone number to which SMS notifications are sent. The phone number can contain digits (0-9), plus sign (+), minus sign (-), and space (␣) characters only.
userName	Optional string. Name of the user. If no user name is provided during subscription, an anonymous subscription is created by using the phone number as the user name. If a user name is provided during subscription, it must also be provided during unsubscription.
appId	Mandatory string for subscribe. The ID of the application that contains the SMS gateway definition. The application ID is constructed from the application name, application environment, and application version. For example, version 1.0 of Android application SMSPushApp has **appId** = SMSPushApp-android-1.0.

Note: If any parameter value contains special characters, this parameter must be encoded by using URL encoding, also known as percent encoding, before the URL is constructed. Parameter values containing only the following characters do not need to be encoded:

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>
```

Sending push notifications to the device

IBM Worklight provides a unified push notifications API to enable you to send push notifications from the server to the device.

About this task

To start receiving push notifications, an application must first subscribe to a push notification event source. The event source is a push notification channel to which mobile applications can register. An event source is defined within a Worklight adapter. The device user must approve the push notification subscription.

When the user approves, the device registers with an Apple, Google, or Microsoft push server to obtain a device token that is used to identify the device. The device then sends a subscription request to the Worklight Server. This operation is performed automatically within IBM Worklight.

The unified push notifications API comprises two elements:

- The Adapter API, to manage subscriptions, push and pull notifications from the server, and to send push notifications to devices.
- The Application API, to subscribe to and unsubscribe from push notification event sources, and to handle incoming notifications.

For more information about the API, see WL.Client.Push.

Procedure

1. A notification is retrieved from the back-end system. Notifications can be polled from the back-end system, or the back-end system can explicitly push a new notification.
2. The notification is retrieved by the adapter, processed, and then sent through the corresponding push service mediator (Apple, Google, or Microsoft). For more information about setting up push notifications, see Setting up push notifications.
3. The push service mediator receives the notification and sends it to a device.
4. The device processes the received notification.

Sending SMS push notifications to the device

In addition to standard push notifications, you can also send Short Message Service (SMS) messages, more commonly known as text messages, to user devices.

The SMS notification framework extends the push notification framework. In order to receive SMS notifications, the user must first subscribe to a push notification event source.

About this task

SMS support is provided for Apple, Google, and Windows Phone 8 devices, and for BlackBerry devices that support SMS functions. IBM Worklight includes the capability to send SMS notifications to all platforms that provide SMS support.

Procedure

1. An SMS notification infrastructure is set up, comprising a Worklight adapter acting as a connector to an app running on a mobile device.
2. The user of the mobile device sends a subscribe request from the application to the event source declared in the Worklight adapter, by using the client-side SMS subscribe API **`WL.Client.Push.subscribeSMS`**.
3. The user subscription to the event source is registered at the Worklight Server.

4. When the back-end service has to notify the user, it invokes a method in the Worklight adapter.
5. The adapter checks whether an SMS subscription already exists for that user and, if it does, sends the SMS alert message through a preconfigured SMS Aggregator.
6. Optional: If SMS notifications are no longer required, you can unsubscribe. The subscription is deleted either by an application that calls the **`WL.Client.Push.unsubscribeSMS`** API, or by using the Admin console. For more information, see Administering push notifications with the Worklight Console.

   For a detailed scenario-based example showing SMS messaging, see the developerWorks article Send SMS push notifications to your mobile app using IBM Worklight.

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.

Configuring a polling event source to send push notifications

Polling event sources can be used to generate notification events, such as push notifications, that the IBM Worklight client framework can subscribe to.

About this task

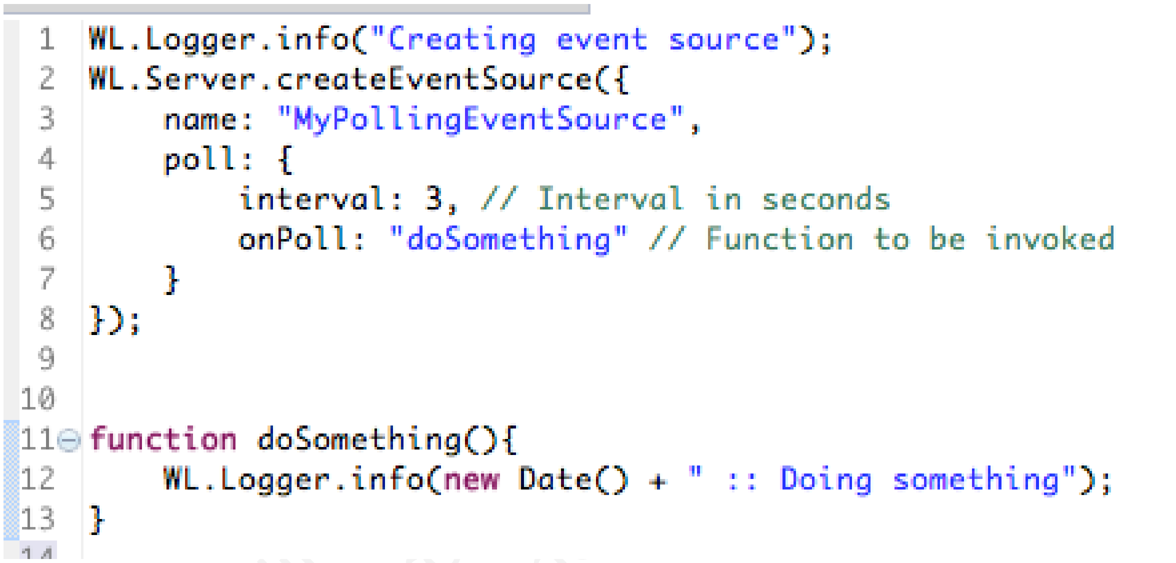
The Worklight adapter framework provides the ability to implement event sources, which can be used to generate notification events such as push notifications. However, notifications must be retrieved from a back-end system before they can be sent out. Event sources can either poll notifications from the back-end system, or wait for the back-end system to explicitly push a new notification.

This task describes how to create a polling event source, and use it to send push notifications. A polling event source is a long-running task that has the following mandatory properties:

- Event source name
- Polling interval
- Polling function

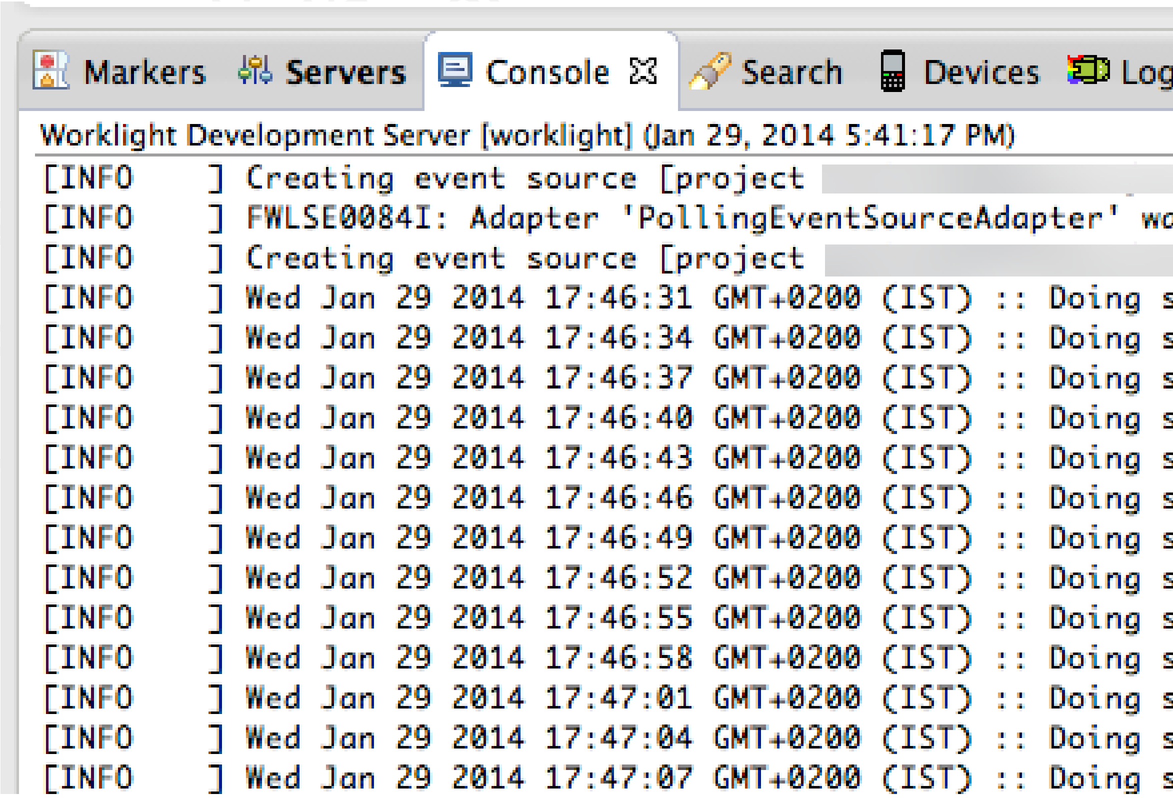
Procedure

- Consider the following simple example. The diagram shows a sample for a basic polling event source:



```
WL.Logger.info("Creating event source");
WL.Server.createEventSource({
    name: "MyPollingEventSource",
    poll: {
        interval: 3, // Interval in seconds
        onPoll: "doSomething" // Function to be invoked
    }
});


function doSomething(){
    WL.Logger.info(new Date() + " :: Doing something");
}
```

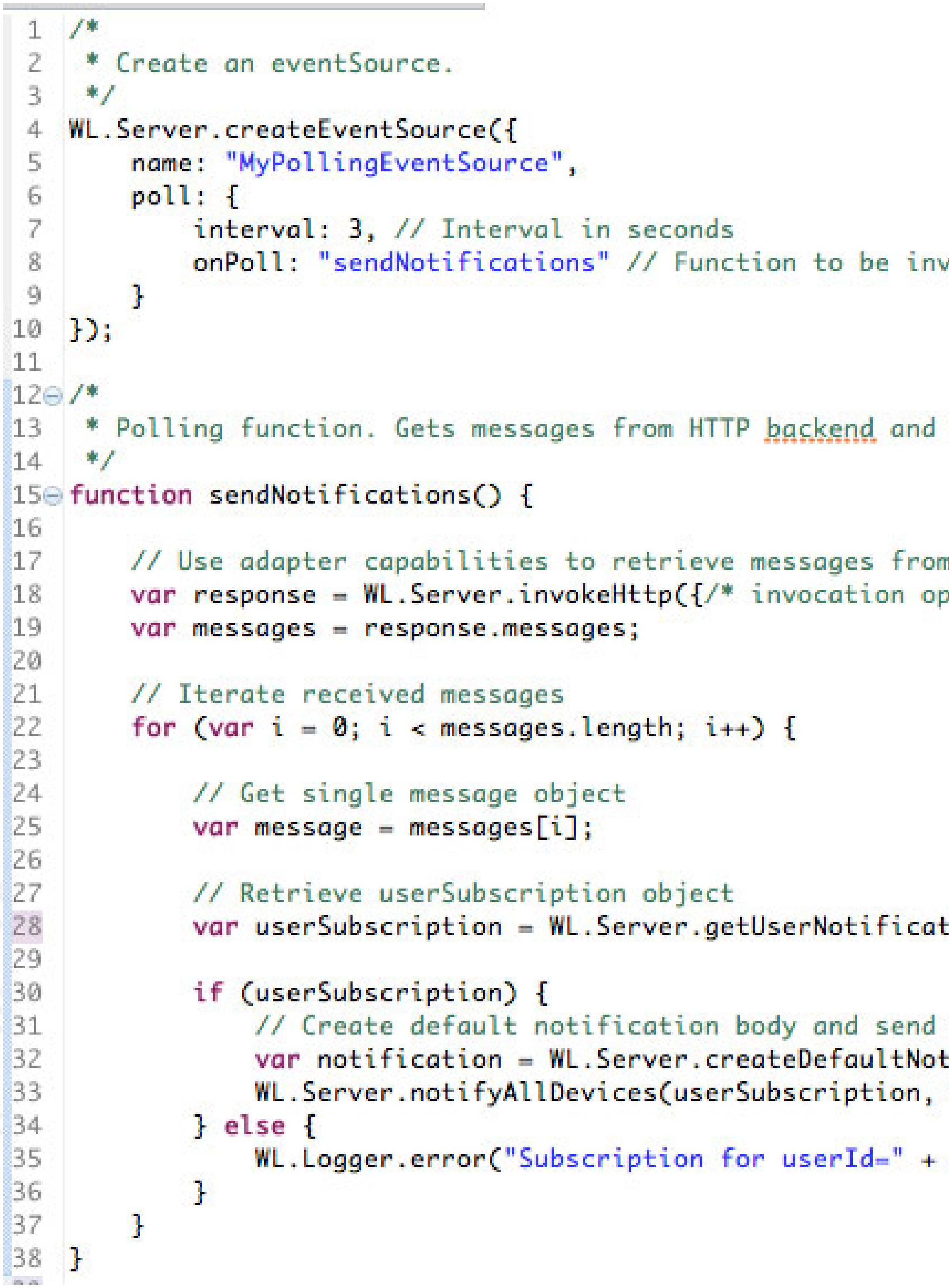
The doSomething() function is invoked every three seconds. If you deploy this adapter to the Worklight Server, you see the following logs in the server console:



```
Worklight Development Server [worklight] (Jan 29, 2014 5:41:17 PM)
[INFO    ] Creating event source [project
[INFO    ] FWLSE0084I: Adapter 'PollingEventSourceAdapter' wo
[INFO    ] Creating event source [project
[INFO    ] Wed Jan 29 2014 17:46:31 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:34 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:37 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:40 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:43 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:46 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:49 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:52 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:55 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:46:58 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:47:01 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:47:04 GMT+0200 (IST) :: Doing s
[INFO    ] Wed Jan 29 2014 17:47:07 GMT+0200 (IST) :: Doing s
```

The log shows that the doSomething() function is invoked at 3-second intervals.

- This second example shows a more realistic example of a polling event source:



```
1  /*
2   * Create an eventSource.
3   */
4  WL.Server.createEventSource({
5       name: "MyPollingEventSource",
6       poll: {
7            interval: 3, // Interval in seconds
8            onPoll: "sendNotifications" // Function to be inv
9       }
10 });
11
12 /*
13  * Polling function. Gets messages from HTTP backend and
14  */
15 function sendNotifications() {
16
17      // Use adapter capabilities to retrieve messages from
18      var response = WL.Server.invokeHttp({/* invocation op
19      var messages = response.messages;
20
21      // Iterate received messages
22      for (var i = 0; i < messages.length; i++) {
23
24           // Get single message object
25           var message = messages[i];
26
27           // Retrieve userSubscription object
28           var userSubscription = WL.Server.getUserNotificat
29
30           if (userSubscription) {
31                // Create default notification body and send
32                var notification = WL.Server.createDefaultNot
33                WL.Server.notifyAllDevices(userSubscription,
34           } else {
35                WL.Logger.error("Subscription for userId=" +
36           }
37      }
38 }
```

The sample includes the following key elements:

- Lines 7 - 8: The polling event source continuously invokes a `sendNotifications()` function with a 3-second interval.
- Lines 18 - 19: Every time the `sendNotifications()` function is invoked it requests messages data from the back-end. The sample shows an HTTP back-end, but it could be any other type of back-end Worklight adapter support, for example, SQL. The code assumes that the following JSON

markup is returned by the back-end. However, since the Worklight adapter knows how to automatically convert data to JSON, the back-end data could also be XML.

```
{
  messages: [{
    userId: "John",
    text: "New incoming transition",
    badge: 2,
    payload: {}
  },
  {
    userId: "Bob",
    text: "Please approve withdrawal",
    badge: 5,
    payload: {}
  }]
}
```

- Line 22: The code iterates over the received messages array.
- Line 25: Every message contains the user ID of a user that the notification should be sent to.
- Line 28: Using this user ID, the code tries to obtain a `userSubscription` object.
- Lines 30 - 33: If a `userSubscription` object is found for the specified user ID, a new notification is created and is sent to all user devices.
- Line 35. If a `userSubscription` object is not found for the specified user ID, an error is logged.

An important feature of a polling event source is that unlike regular adapter procedures, the polling function is triggered by the Worklight Server itself, and not by the incoming request. Therefore any data or APIs related to request or session context are not available or functional. For example, APIs such as WL.Server.getActiveUser() or WL.Server.getClientRequest() are not functional. Also, you do not need to expose polling function in the adapter XML file.

Using two-way SMS communication

SMS two-way communication enables communication between a mobile phone and the Worklight Server, over an SMS channel. SMS messages that originate from the mobile device can be sent to the Worklight Server through an external SMS gateway. The Worklight Server can then send a response message back to the originating mobile device.

Before you begin

To run SMS two-way communication, the mobile device must support SMS functions.

About this task

Keywords or shortcodes should be configured with the third-party SMS gateway. The gateway should be configured to forward SMS messages to the SMS servlet of the Worklight Server, either directly or through a reverse proxy URL, based on the topology in your environment:

```
http://hostname:port/context/receiveSMS
```

The SMS messages that are sent from mobile phones are forwarded to an adapter procedure on the Worklight Server, which is configured by the developer. The adapter procedure can include the logic to process the request, or the procedure can forward the request to a back-end system for processing. The response is returned by using the IBM Worklight notification framework. For more information, see Push notification.

The two-way SMS architecture is summarized in the following figure:

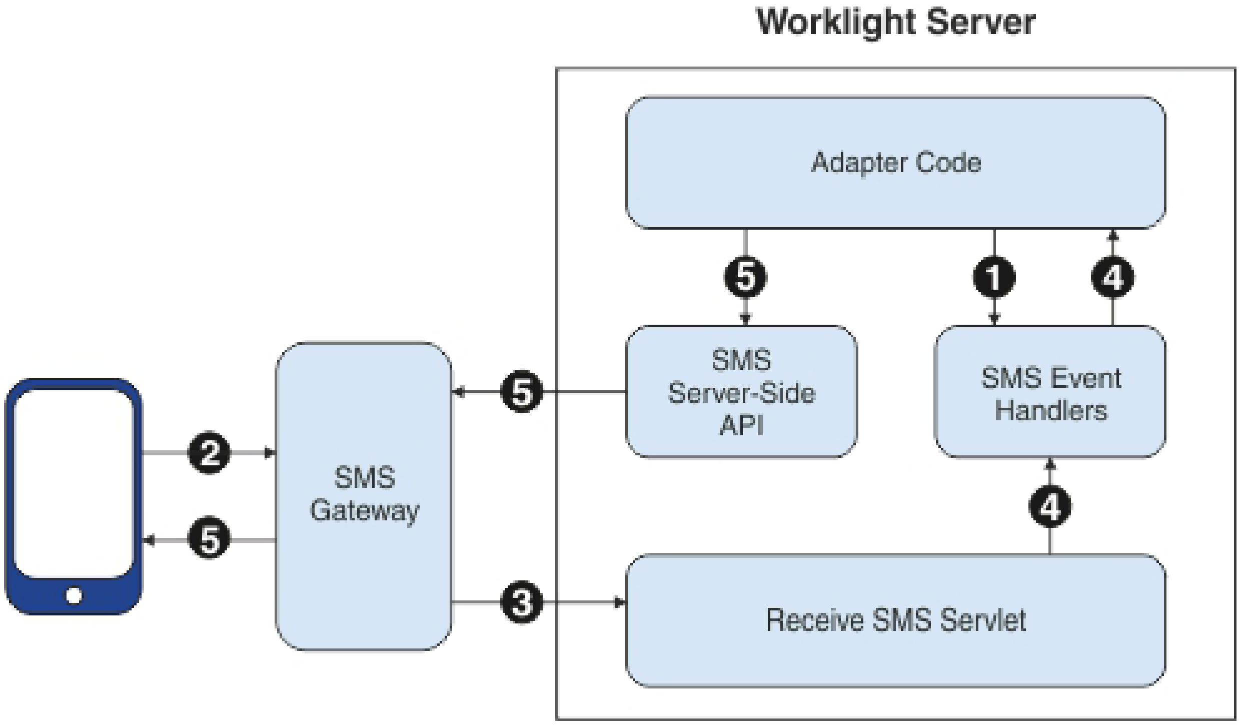


Figure 86. Two-way SMS architecture

1. The adapter registers SMS event handlers on the Worklight Server.
2. SMS messages are sent from mobile devices to the SMS gateway, which is configured with an SMS servlet of Worklight Server.
3. The SMS gateway forwards SMS messages to a configured Worklight URL.
4. An SMS servlet on Worklight Server matches the parameters with filters that are defined in SMS event handlers, and calls an adapter callback procedure.
5. The adapter processes SMS messages and sends an SMS message to the mobile device by using the SMS API.

You use a series of server API methods to send and receive SMS messages:

`WL.Server.createSMSEventHandler`
Create an SMS event handler.

`WL.Server.setEventHandlers`
Set event handlers to implement callbacks for received events.

`WL.Server.subscribeSMS`
Subscribe a phone number to the specified event source.

`WL.Server.unsubscribeSMS`
Unsubscribe the phone number from the specified event source.

`WL.Server.getSMSSubscription`
Return an SMS subscription object for a phone number.

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.

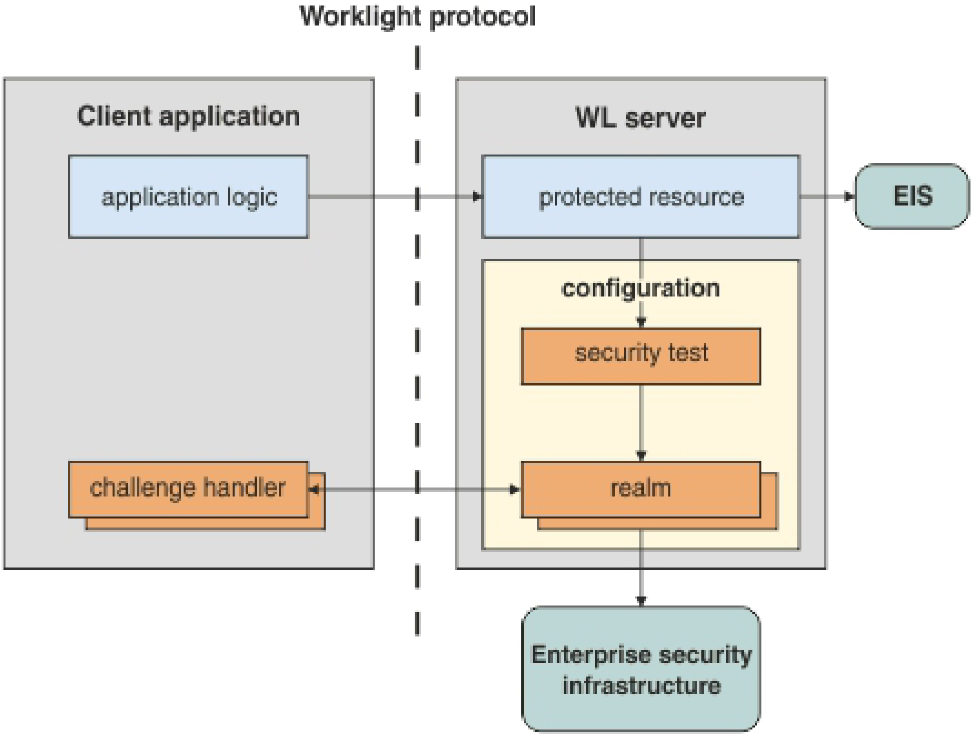
It is key to the success of the application that the Worklight security framework does not include its own user registry, credentials storage, or access control management. Instead, it delegates all those functions to the existing enterprise security infrastructure. This delegation allows Worklight Server 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 that 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 maximizes 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 IBM Security Access Manager 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, it 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 single sign-on (SSO) feature because the user credentials are requested by the gateway.

Further information on security, as it is implemented in IBM Worklight, is provided in the following overview of security features. There are links to the relevant sections of the documentation, which pertain to them.

Integration with IBM Security Access Manager

IBM Security Access Manager can be integrated with IBM Worklight to provide the following protections by using risk-based access decisions to protect Worklight applications and adapters as listed here:

- user authentication
- SSO
- Identity attributes
- Fine-grained authorization

SSO can be achieved to the mobile client and in adapter server connections. The context-based access policies can be defined to provide identity assurance and strong authentication with a one time password (OTP) for adapter-based transactions in Worklight and application authentication.

For more information about IBM Security Access Manager, see IBM Security Access Manager for IBM Worklight.

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 to remotely disable, from the Worklight Console, 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.

For a webSecurityTest:

```
<webSecurityTest name="webTest">
  <testUser realm="wl_anonymousUserRealm"/>
</webSecurityTest>
```

The equivalent as a customSecurityTest:

```
<customSecurityTest name="webTest">
  <test realm="wl_antiXSRFRealm" />
  <test realm="wl_anonymousUserRealm" isInternalUserId="true" />
</customSecurityTest>
```

For a mobileSecurityTest:

```
<mobileSecurityTest name="mobileTest">
  <testUser realm="wl_anonymousUserRealm"/>
  <testDeviceId provisioningType="none" />
</mobileSecurityTest>
```

The equivalent as a customSecurityTest:

```
<customSecurityTest name="mobileTest">
  <test realm="wl_antiXSRFRealm" />
  <test realm="wl_remoteDisableRealm" />
  <test realm="wl_anonymousUserRealm" isInternalUserID="true" />
  <test realm="wl_deviceNoProvisioningRealm" isInternalUserID="true" />
</customSecurityTest>
```

Usually, you add your own realm to your configuration to authenticate users. The following example shows a configuration where the realm named MyUserAuthRealm is the realm that the developer added.

Example with your own realm name as a realm definition for testUser:

For a webSecurityTest:

```
<webSecurityTest name="webTest">
  <testUser realm="MyUserAuthRealm"/>
</webSecurityTest>
```

The equivalent as a customSecurityTest

```
<customSecurityTest name="webTest">
  <test realm="wl_antiXSRFRealm" />
  <test realm="MyUserAuthRealm" isInternalUserId="true" />
</customSecurityTest>
```

For a mobileSecurityTest:

```
<mobileSecurityTest name="mobileTest">
  <testUser realm="MyUserAuthRealm"/>
  <testDeviceId provisioningType="none" />
</mobileSecurityTest>
```

The equivalent as a customSecurityTest:

```
<customSecurityTest name="mobileTest">
  <test realm="wl_antiXSRFRealm" />
  <test realm="wl_remoteDisableRealm" />
  <test realm="MyUserAuthRealm" isInternalUserID="true" />
  <test realm="wl_deviceNoProvisioningRealm" isInternalUserID="true" />
</customSecurityTest>
```

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 recognizes 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 98. Predefined realms: properties of the **<test realm>** element.

Authenticator class name	Login module reference	Description
wl_antiXSRFRealm	**WLAntiXSRFLoginModule**	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.
wl_deviceNoProvisioningRealm	**WLDeviceNoProvisioningLoginModule**	A default *device identity* 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.
wl_deviceAutoProvisioningRealm	**WLDeviceAutoProvisioningLoginModule**	The description of this parameter is the same as for **wl_deviceNoProvisioningRealm**, but the obtained device identity is automatically provisioned by the Worklight Server. This realm must be used with **wl_authenticityRealm**.
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
  <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 Console, usage reports, and the Application Center console.

The Worklight web applications that require authentication are the Worklight Console, the IBM Worklight usage reports, and the Application Center console. The Worklight Console and Worklight usage reports are configured by using `<resource>` elements in the `authenticationConfig.xml` file.

The 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 138.

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 99. 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 100. 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
`login-page`	Path to a user-defined login page template, relative to the web application context under the `conf` directory. A sample `login.html` template file is provided under this directory when creating a Worklight project in Worklight Studio. The authenticator renders the login page template with the error messages. To display the error message, use the placeholder `${errorMessage}` within your login page template, as depicted in the example.
`auth-redirect`	Path to a user defined login page (`html`/`jsp`) 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
`<cookie-name>`	Optional. The name of the persistent cookie. If this parameter is not specified, the default name, `WL_PERSISTENT_COOKIE`, is used.

```
<realm name="PersistentCookie" loginModule="dummy">
<className> com.worklight.core.auth.ext.PersistentCookieAuthenticator </className>
</realm>
```

Adapter-based authentication

The adapter authenticator enables you to develop custom authentication logic using JavaScript. When you use adapter-based authentication, the entire authentication logic (the credentials validation and the creation of a user identity) is implemented via a JavaScript function within an IBM Worklight adapter.

Since all of the validation logic that is usually done in a login module is now performed in the adapter's JavaScript code (and no further validation is required), adapter-based authentication should only be used with a `NonValidatingLoginModule` (see “Non-validating login module” on page 617).

The adapter authenticator fully qualified Java class name is `com.worklight.integration.auth.AdapterAuthenticator`, and it has two parameters:

- **`login-function`** (mandatory)
- **`logout-function`** (optional)

Both parameters specify adapter function names. The syntax is "*adapter-name.function-name*"; for example, "`myAuthAdapter.onAuthRequired`".

Example of configuration of adapter-based authentication in the `authenticationConfig.xml` file

```
<securityTests>
  <customSecurityTest name="AuthenticationAdapter-securityTest">
    <test isInternalUserID="true" realm="AdapterAuthRealm"/>
  </customSecurityTest>
</securityTests>

<realms>
  <realm loginModule="AdapterAuthLoginModule" name="AdapterAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="AuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="AuthAdapter.onLogout"/>
  </realm>
</realms>
```

```
<loginModules>
  <loginModule name="AdapterAuthLoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
  </loginModule>
</loginModules>
```

You will now need to implement the login function (`AuthAdapter.onAuthRequired` in the example above) and logout function (`AuthAdapter.onLogout` in the example above) in your `adapter.js` source file.

Note that both **`login-function`** and **`logout-function`** should only be used internally by a Worklight Server. For this reason, it is important that you do not expose them as procedures in the `adapter .xml` file. The function that receives credentials, on the other hand, is directly invoked by a client, and so you need to expose it in the `adapter xml` file. Keep in mind that this function must be accessed by unauthenticated clients, and so it should not be protected by a security test.

In addition to implementing **`login-function`** and **`logout-function`** described above, you also need to implement an adapter function that will receive credentials from the client, validate them, and create a user identity; for example, function `submitCredentials (`*`user`*`, `*`password`*`)`.

The login function

The **`login-function`** parameter specifies the name of the JavaScript function to be invoked once the login process is triggered. The triggering can happen when either the client application explicitly invokes the `WL.Client.login()` API, or when an unauthenticated attempt to access a resource protected by the adapter authentication realm is made. Use this function to return a payload to the client notifying it about the required authentication. The **`login-function`** receives original request headers converted to JSON as a first function argument so that they can be used to decide on the kind of authentication needed, for example. The response from **`login-function`** will be returned to the client instead of the expected one.

The submit credentials function

This is the function that actually performs the authentication. The client should call this function with arguments containing user credentials or authentication data. It should then validate the credentials and once validated, this function should use `WL.Server.setActiveUser(`*`realm`*`, `*`identity`*`)` to register the authenticated identity. The function can include a flag or message in the response to let the application know if the login was successful or not.

The logout function

The **`logout-function`** parameter specifies the name of the JavaScript function to be invoked once logout from the realm has occurred. The logout can be triggered by having the client application call the `WL.Client.logout()` API, or when the Worklight Server decides to invalidate the session (for example, a session timeout). The **`logout-function`** receives no arguments. This function should use the `WL.Server.seActiveUser(`*`realm`*`, null)` to remove the authenticated identity. Note that the same `WL.Server` function is used to login but here `null` is used instead of a `userIdentity` object to do the logout and remove the authenticated identity.

Further information

For an example of how to correctly implement adapter authentication and the adapter functions required, see the module *Adapter-based authentication* under category 8, *Authentication and security*, in Chapter 3, "Tutorials and samples," on page 27.

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
`login-page`	Mandatory. The login page URL relative to the web application context.
`error-page`	Optional. The error page URL relative to the web application context. If this parameter is not set, the URL from the `login-page` is also used for the `error-page`.
`cookie-domain`	Optional. A String such as `example.com`, which specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The single sign-on is then restricted to the application server host name and does not work with other hosts in the same domain. **Note:** This parameter is deprecated. Define this parameter in the `<loginModule>` entry for the `WebSphereLoginModule` instead. If the parameter is defined in both places, the value in the `<loginModule>` entry takes precedence.
`httponly-cookie`	Optional. A String with a value of either `true` or `false`, which specifies whether the cookie has the HttpOnly attribute set. This attribute helps to prevent cross-site scripting attacks. **Note:** This parameter is deprecated. Define this parameter in the `<loginModule>` entry for the `WebSphereLoginModule` instead. If the parameter is defined in both places, the value in the `<loginModule>` entry takes precedence.

Parameter	Description
cookie-name	Optional. A String that specifies the name of the LTPA SSO cookie. If this parameter is not set, the default cookie name is LtpaToken. **Note:** This parameter is deprecated. Define this parameter in the <loginModule> entry for the WebSphereLoginModule instead. If the parameter is defined in both places, the value in the <loginModule> entry takes precedence.

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"/>
</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.
audit	Optional. Defines whether login attempts that use the login module are logged in the audit log. The log file is *Worklight Project Name*/server/log/audit/audit.log. Valid values are: **true** Login and logout attempts are logged in the audit log. **false** Default. Login and logout attempts are not logged in the audit log.

The <loginModule> element has the following subelements:

Element	Description
<className>	Mandatory. The class name of the login module. For details of the supported login modules, see the following topics.
<parameter>	Optional. An initialization property of the login module. The supported properties and their semantics depend on the login module class. This element can occur multiple times.

Non-validating login module

The non-validating login module accepts any user name and password passed by the authenticator.

Class Name

```
com.worklight.core.auth.ext.NonValidatingLoginModule
```

Parameters

None

```
<loginModule name="dummy" canBeResourceLogin="false" isIdentityAssociationKey="true">
<className> com.worklight.core.auth.ext.NonValidatingLoginModule </className>
</loginModule>
```

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
console.username	Name of the user who can access the Console
console.password	Password of the user who can access the Console. The password can be encrypted as indicated in "Storing properties in encrypted format" on page 779.

```
<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

The login module class has the following parameters:

Parameter	Description
cookie-domain	Optional. A String such as example.com, which specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The single sign-on is then restricted to the application server host name and does not work with other hosts in the same domain.
httponly-cookie	Optional. A String with a value of either true or false, which specifies whether the cookie has the HttpOnly attribute set. This attribute helps to prevent cross-site scripting attacks.
cookie-name	Optional. A String that specifies the name of the LTPA SSO cookie. If this parameter is not set, the default cookie name is LtpaToken.
role	Optional. A String that specifies the Java EE role that the authenticated user must belong to for the login to be successful. If the parameter is not specified, no role checking is performed.

Note: When you specify a role parameter, the role must be defined in the Worklight web application deployment descriptor (`web.xml`). A set of users or groups must be mapped to that role by using the usual WebSphere Application Server mechanisms.

```
<loginModule name="WASLTPAModule">
  <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
  <parameter name="role" value="wluser"/>
  <parameter name="cookie-domain" value="example.com"/>
  <parameter name="httponly-cookie" value="true"/>
  <parameter name="cookie-name" value="LtpaToken2"/>
</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 *john*, and others require the user name and the domain, for example *john@server.com*. 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`

Parameter	Description	Sample values
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={user 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="(&amp;(objectClass=user)(sAMAccountName={username
  <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 authentication

You can require mobile devices to authenticate themselves. Device identity is used in several places within IBM Worklight. You can use provisioning, which is the process of obtaining a security certificate. There are three modes of the provisioning process.

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:
  - 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.
  - The unique device ID is stored in the device keychain.
- On Android:
  - To calculate the unique device ID, device properties are used, such as the WiFi Mac address. This guarantees the uniqueness of the device ID, and make the process more secure by generating the device ID at the start of each application.
  - The unique device ID can be unique either to the application or to all applications from the same vendor.
  - 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 968.

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 27.

Scope of mobile device authentication

In addition to requiring users to authenticate before they access certain resources, you can also require mobile devices to authenticate before apps installed on them can access the Worklight Server.

Device and application authentication is a process that allows making claims of type "this is application A installed on device D".

Device and application authentication is relevant only for applications that are installed on mobile devices.

Mobile device provisioning

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 usually suitable 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.

Configuring and implementing device provisioning

You can change the default behavior with regard to granularity, pre-required realms, and CA certificates. You can also implement custom provisioning.

Procedure

1. To change the default behavior of provisioning granularity and pre-required realms, define a new realm for device provisioning and add the following `<realm>` element to the `<realms>` element in the `authenicationConfig.xml` file. Then, use it in your security test of choice:

```
<realm name="wl_myProvisioningRealm"
       loginModule="WLDeviceAutoProvisioningLoginModule">
  <className>com.worklight.core.auth.ext.DeviceAutoProvisioningAuthenticator</className>
  <parameter name="provisioned-entity" value="application" />
  <parameter name="pre-required-realms" value="wl_authenticityRealm" />
<realm>
```

   where *provisioned-entity* can have one of the following values:
   - application
   - device
   - group:<*group-name*>, where *group-name* is the name of the provisioning application group

   and *pre-required-realms* is a comma-separated list of realm names that are required to be successfully logged in to before provisioning is allowed to begin.

   Note: Applications must be signed by the same signing credentials and (on iOS) share the same `bundleID` prefix.

2. 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 772

   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`.

3. 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, *Authentication and security*, in Chapter 3, “Tutorials and samples,” on page 27.

Implementing client-side components for custom device provisioning

You can implement client-side components for custom device provisioning.

Before you begin

The following prerequisites are required for device provisioning:

- Worklight Studio and Worklight Server, from IBM Worklight Enterprise Edition or IBM Worklight Consumer Edition.
- Android apps must be built for production and signed by a certificate other than the included debugging certificate.
- In the Application Center console, app authentication must be set to `enabled, blocking`.

The included Worklight Development Server can be used for device provisioning.

About this task

To implement client-side components for custom device provisioning, complete the following steps.

Procedure

1. Create an application.
2. Add an iPhone, iPad, or Android environment to the application.
3. Configure the application for the Application Authenticity test. The authenticity test works only IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition. For more information about application authenticity, see “IBM Worklight application authenticity overview” on page 631.
4. Update the application HTML file.

```
<body id="content" style="display: none;">
  <div id="AppBody">
    <div class="header">
      <h1>CustomProvisioningApp</h1>
    </div>
    <div id="wrapper">
      Device authentication with custom device provisioning was not complete
    </div>
    <button id="conectToServerButton">
      Connect to Worklight server
    </button>
  </div>
  <div id="ProvBody" style="diplay: none">
    <div id="provisioningError></div>
    <input id="submitProvCodeButton">Send</button>
  </div>
...
</body>
```

5. Add a listener to `connectToServerButton`.

6. Optional: Use the WL.Client.connect() API to connect to the Worklight Server if the connectOnStartup property is set to false in the initOptions.js file.

```
function wlCommonInit() {
    $("#connectToServerButton").click(function(){
        WL.Client.connect();
    });
}
```

7. For the WL.Client.connect() function to trigger authentication, specify the Worklight Server as the protected resource by adding a custom security test or mobile security test in the application descriptor.

```
<iphone securityTest="ADPSecurityTest" version="1.0">
```

or

```
<ipad securityTest="ADPSecurityTest" version="1.0">
```

or

```
<android securityTest="ADPSecurityTest" version="1.0">
```

8. Add a CustomDeviceProvisioningRealmChallengeHandler.js file, and reference it from the main HTML file.

9. Implement the following methods that are required by the device provisioning challenge handler:
   - handler.createCustomCsr(challenge) - This method is responsible for returning custom properties that are added to CSR. Add a custom activationCode property, which is used in the adapter's validateCSR function.

     Note: This method is asynchronous to allow collecting custom properties through native code or separate flow.
   - handler.processSuccess(identity) - This method is invoked when certificate validation is successfully completed by using the validateCertificate adapter function.
   - handler.handleFailure() - This method is invoked when certificate validation fails.

10. Implement the device provisioning challenge handler.

```
var customDevProvChallengeHandler =
    WL.Client.createProvisioningChallengeHandler("CustomDeviceProvisioningRealm");

customDevProvChallengeHandler.createCustomCsr = function(challenge) {
    WL.Logger.debug("createCustomCsr :: " + JSON.stringify(challenge));

    $("#AppBody").hide();
    $("#ProvBody").show();
    $("#provisioningCode").val("");

    if (challenge.error) {
        $("#provigioningError").html(new Date() + " " + challenge.error);
    } else {
        $("#provisioningError").html(new Data() + " Enter activation code.");
    }

    $("#submitProvCodeButton").click(function() {
        var customCsrProperties = {
            activationCode: $("#provisioningCode").val()
        };
        customDevProvChallengeHandler.submitCustomCsr(customCsrProperties, challenge);
    });
};
```

```
customDevProvChallengeHandler.processSuccess = function(identity) {
    WL.Logger.debug("processSuccess :: " + JSON.stringify(identity));
    $("#connectToServerButton").hide();
    $("#AppBody").show();
    $("#ProvBody").hide();
    $("#wrapper").text("Device authentication with custom device provisioning " +
        "was successfully completed");
};

customDevProvChallengeHandler.handleFailure = function() {
    WL.Logger.debug("handleFailure");
    $("#AppBody").show();
    $("#ProvBody").hide();
    $("#wrapper").text.("Server has rejected your device. You must reinstall the
        application and perform device provisioning again.");
};
```

Results

You implemented client-side components for custom device provisioning.

What to do next

You can implement server-side components for custom device provisioning. For more information about implementing server-side components, see "Implementing server-side components for custom device provisioning." For more information about custom device provisioning, see the module *Custom device provisioning* under category 8, *Authentication and security*, in Chapter 3, "Tutorials and samples," on page 27.

Implementing server-side components for custom device provisioning

You can implement server-side components for custom device provisioning.

About this task

To implement server-side components for custom device provisioning, complete the following steps.

Procedure

1. Create an adapter named `ProvisioningAdapter`.
2. Add two functions with the following signatures to the adapter's JavaScript file:
   - `validateCSR(clientDN, csrContent)` - This function is invoked only during initial device provisioning. The function is used to check whether the device is authorized to be provisioned. After the device is provisioned, this function is not invoked again.
   - `validateCertificate(certificate, customAttributes)` - This function is invoked each time that the mobile application establishes a new session with the Worklight Server. The function is used to validate that the certificate that the application or device possesses is still valid and that the application or device is allowed to communicate with the Worklight Server.

   Note: These functions are called internally by the Worklight authentication framework. Do not declare them in the adapter's XML file.
3. Configure the `authenticationConfig.xml` file.

a. Add a realm named CustomDeviceProvisioningRealm to the authenticationConfig.xml file.
   - Use CustomDeviceProvisioningLoginModule for the loginModule.
   - Use the auto provisioning authenticator className parameter.
   - Add a validate-csr-function parameter.
   - The value of this parameter points to an Adapter function that performs the certificate signing request (CSR) validation.

```
<realms>
    <realm name="CustomDeviceProvisioningRealm"
           loginModule="CustomDeviceProvisioningLoginModule">
        <className>com.worklight.core.auth.ext.DeviceAutoProvisioningAuthenticator</className>
        <parameter name="validate-csr-function"
                   value="ProvisioningAdapter.validateCSR" />
    </realm>
</realms>
```

b. Add the loginModule named CustomDeviceProvisioningLoginModule.
   - Use the auto provisioning login module className parameter.
   - Add a validate-certificate-function parameter.
   - The value of this parameter points to an Adapter function that performs certificate validation.

```
<loginModules>
    <loginModule name="CustomDeviceProvisioningModule">
        <className>com.worklight.core.auth.ext.DeviceAutoProvisioningLoginModule</classname>
        <parameter name="validate-certificate-function"
                   value="ProvisioningAdapter.validateCertificate" />
    </loginModule>
</loginModules>
```

c. Create a securityTest named mobileSecurityTest.
   - Add a mandatory <testAppAuthenticity /> test.
   - Add a mandatory <testDeviceId /> test.
   - Specify provisioningType="custom".
   - Specify realm="CustomDeviceProvisioningRealm".

```
<securityTests>
    <mobileSecurityTest name="CustomDeviceProvisioningSecurityTest">
        <testAppAuthenticity />
        <testDeviceId provisioningType="custom" realm="CustomDeviceProvisioningRealm" />
    </mobileSecurityTest>
</securityTests>
```

Results

You implemented server-side components for custom device provisioning.

Example

The following example shows the validateCSR function:

```
function validateCSR(clientDN, csrContent) {
    WL.Logger.log("validateCSR :: clientDN :: " + JSON.stringify(clientDN));
    WL.Logger.log("validateCSR :: csrContent :: " + JSON.stringify(csrContent));

    var activationCode = csrContent.activationCode;

    // This is a place to perform validation of csrContent and update clientDN if required.
    // You can do it using adapter backend connectivity
    if (activationCode == "worklight") {
        response = {
            isSuccessful: true,
            clientDN: clientDN + ",CN=someCustomData",
            attributes: {
                customAttribute: "some-custom-attribute"
            }
        };
    } else {
        response = {
            isSuccessful: false,
            errors: ["Invalid activation code"]
        };
    }

    return response;
}
```

The following example shows the `validateCertificate` function:

```
function validateCertificate(certificate, customAttributes) {
    WL.Logger.log("validateCertificate :: certificate :: + "JSON.stringify(certificate));
    WL.Logger.log("validateCertificate :: customAttributes :: + "JSON.stringify(customAttributes))

    // Additional custom certificate validations can be performed here.

    return {
        isSuccessful: true
    };
}
```

What to do next

You can implement client-side components for custom device provisioning. For more information about implementing client-side components, see “Implementing client-side components for custom device provisioning” on page 625. For more information about custom device provisioning, see the module *Custom device provisioning* under category 8, *Authentication and security*, in Chapter 3, “Tutorials and samples,” on page 27.

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

- 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 772.

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 in an application authenticity security test. Most IBM 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)

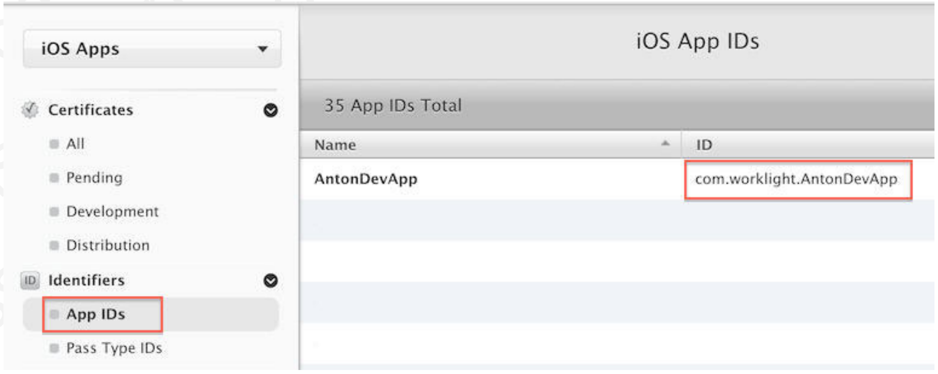
Currently, application authenticity is supported only on iOS and Android platforms.

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.
2. 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:



Name	ID
AntonDevApp	com.worklight.AntonDevApp

   In addition, for a native iOS app, in your XCode project, under **Build Settings** > **Linking** > **Other Linker Flags**, add the flag `-ObjC`.
   - 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).

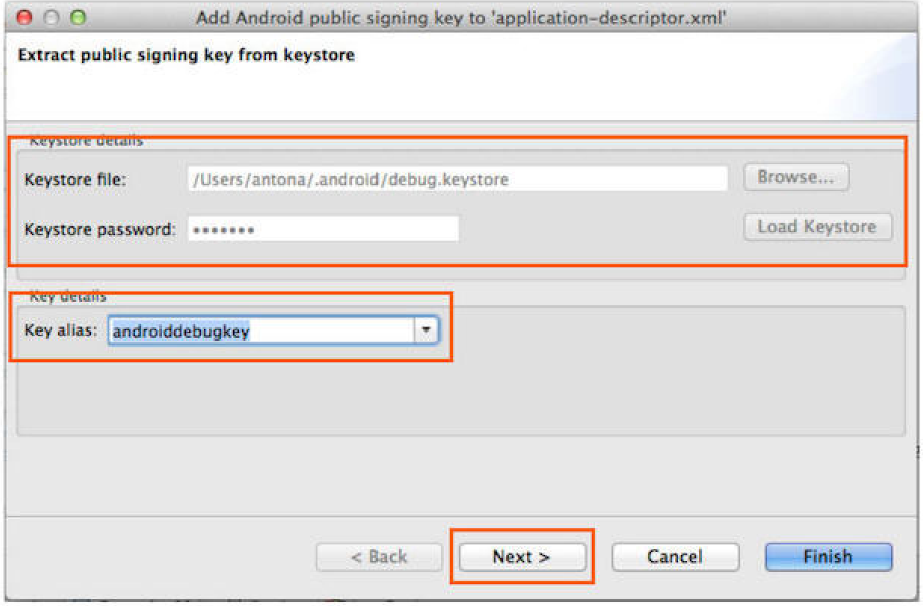
     Worklight Studio provides tools to simplify this process. If you are building your application for distribution (production), you must extract the public key from the certificate you are using to sign your production-ready application. If you are building your application in a development environment, 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:

a. Right-click your Android environment and select **Extract public signing key**.

b. Specify the location and password of the keystore file and click **Load Keystore**.

c. The default password for `debug.keystore` is **`android`**. Select key alias and click **Next**.



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

Extract public signing key from keystore

Keystore details

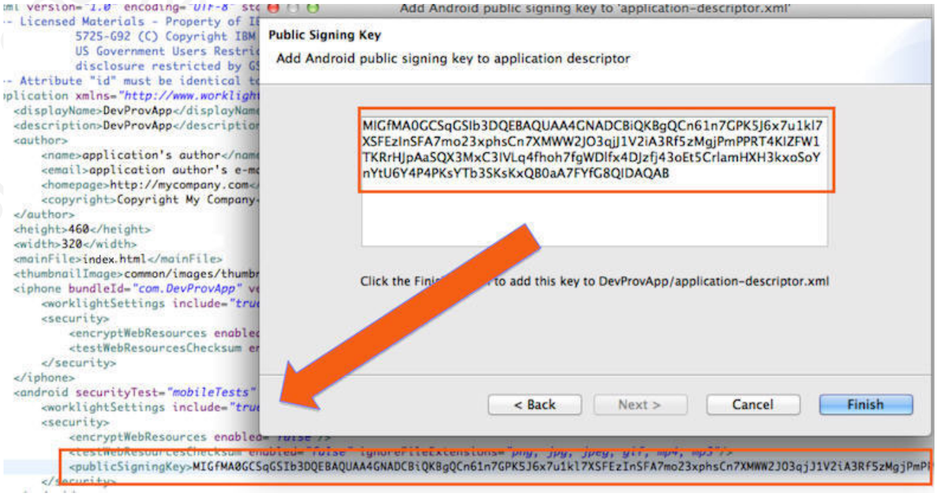
Keystore file: /Users/antona/.android/debug.keystore

Keystore password: •••••••

Key details

Key alias: androiddebugkey

- The public key is displayed on a window:



```
ml version="1.0" encoding="UTF-8" st
-- Licensed Materials - Property of I
           5725-G92 (C) Copyright IBM
           US Government Users Restri
           disclosure restricted by G
-- Attribute "id" must be identical t
plication xmlns="http://www.worklight
  <displayName>DevProvApp</displayName
  <description>DevProvApp</descriptio
  <author>
      <name>application's author</nam
      <email>application author's e-m
      <homepage>http://mycompany.com<
      <copyright>Copyright My Company
  </author>
  <height>460</height>
  <width>320</width>
  <mainFile>index.html</mainFile>
  <thumbnailImage>common/images/thumb
  <iphone bundleId="com.DevProvApp" v
      <worklightSettings include="tru
      <security>
          <encryptWebResources enable
          <testWebResourcesChecksum e
      </security>
  </iphone>
  <android securityTest="mobileTests"
      <worklightSettings include="tru
      <security>
          <encryptWebResources enabled="false"/>
          <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
          <publicSigningKey>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCn61n7GPK5J6x7u1kl7XSFEzInSFA7mo23xphsCn7XMWW2JO3qjJ1V2iA3Rf5zMgjPmP
      </security>
```

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

Public Signing Key

Add Android public signing key to application descriptor

```
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCn61n7GPK5J6x7u1kl7
XSFEzInSFA7mo23xphsCn7XMWW2JO3qjJ1V2iA3Rf5zMgjPmPPRT4KIZFW1
TKRrHJpAaSQX3MxC3IVLq4fhoh7fgWDIfx4DJzfj43oEt5CrlamHXH3kxoSoY
nYtU6Y4P4PKsYTb3SKsKxQB0aA7FYfG8QIDAQAB
```

Click the Fini to add this key to DevProvApp/application-descriptor.xml

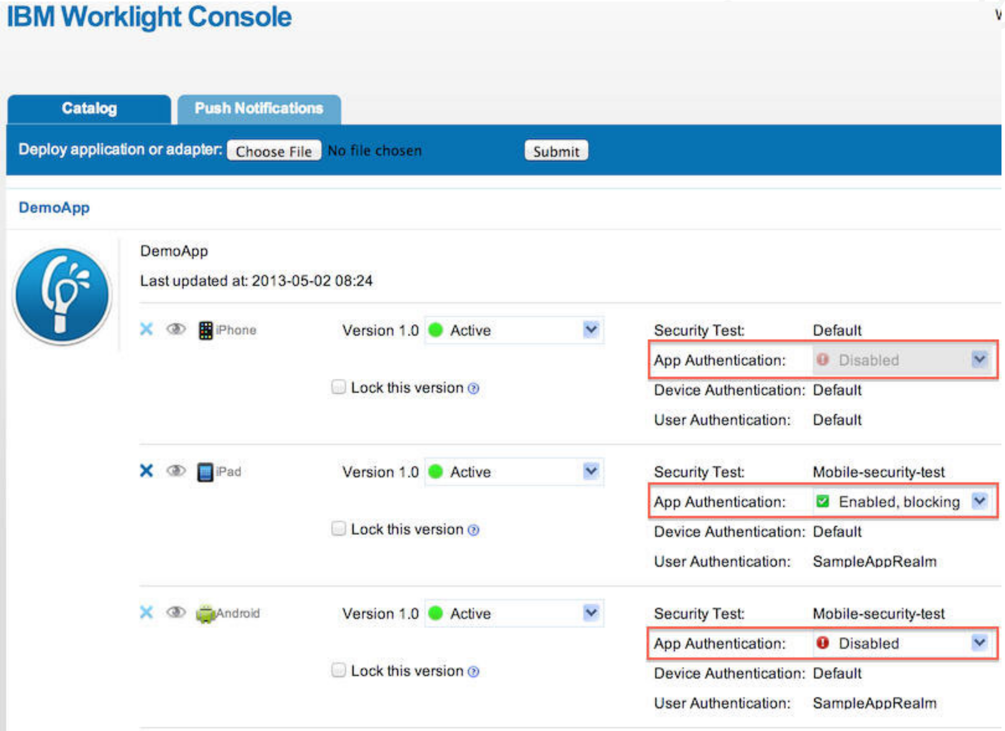
When you click **Finish**, the public key is automatically pasted into the relevant section of `application-descriptor.xml`.

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.



DemoApp

Last updated at: 2013-05-02 08:24

iPhone — Version 1.0 ● Active — □ Lock this version

Security Test:	Default
App Authentication:	Disabled
Device Authentication:	Default
User Authentication:	Default

iPad — Version 1.0 ● Active — □ Lock this version

Security Test:	Mobile-security-test
App Authentication:	✓ Enabled, blocking
Device Authentication:	Default
User Authentication:	SampleAppRealm

Android — Version 1.0 ● Active — □ Lock this version

Security Test:	Mobile-security-test
App Authentication:	Disabled
Device Authentication:	Default
User Authentication:	SampleAppRealm

User certificate authentication realm

The user certificate authentication realm authenticates the user with X.509 certificates that are generated with the Worklight Server together with your public key infrastructure (PKI).

For more information about this realm and how to set it up, see “User certificate authentication overview” on page 995.

Troubleshooting authenticity problems

Find information to help resolve issues that you might encounter related to authenticity.

Table 101. Authenticity troubleshooting guidelines. This table lists possible problems and actions to take to troubleshoot authenticity problems.

Problem	Actions to take
The client runtime continually makes requests to the Worklight Server and the server responds with the same authentication request.	The `WL.Client.addGlobalHeader("Cookie", "name=value")` API in JavaScript might cause a product client to enter an authentication loop. The product uses cookies internally to maintain session states between the client and the server. The `WL.Client.addGlobalHeader` API replaces the entire 'Cookie' header and destroys the session state. Do not use the `WL.Client.addGlobalHeader("Cookie", "name=value")` API. Instead, use the `document.cookie = "name=value"` API within your JavaScript logic. The `document.cookie` API ensures that the cookie is appended to the existing cookie list instead of replacing all existing cookies.

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

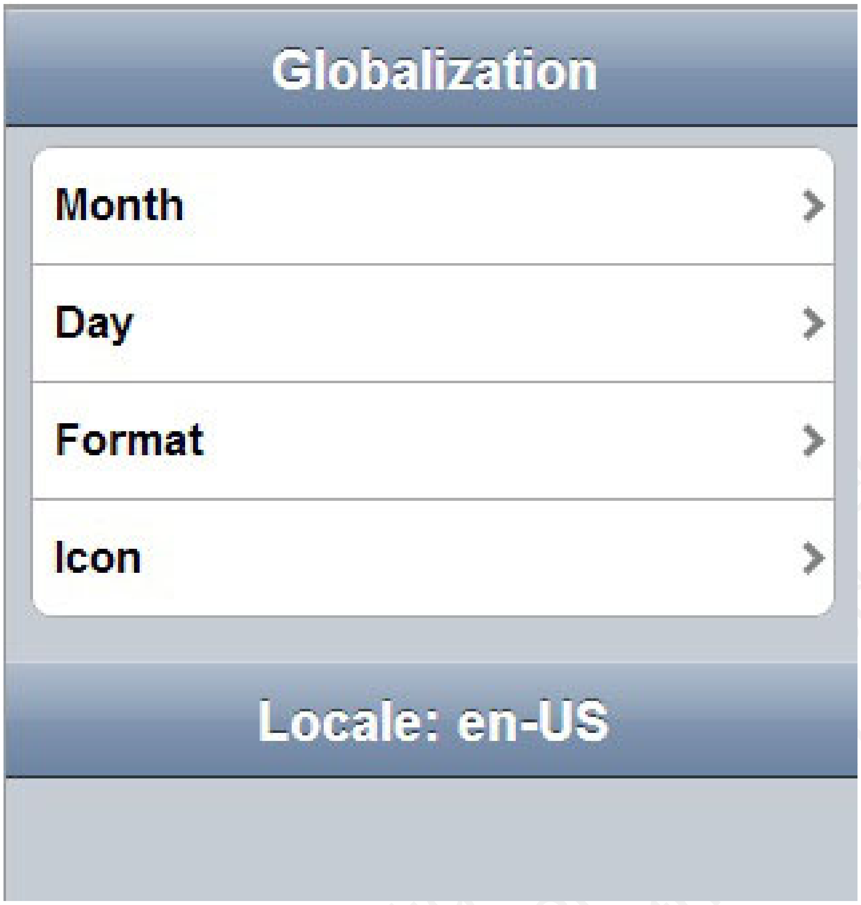
You can use several JavaScript frameworks to globalize your applications: Dojo, jQuery, and Sencha Touch.

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.



Globalization

Month

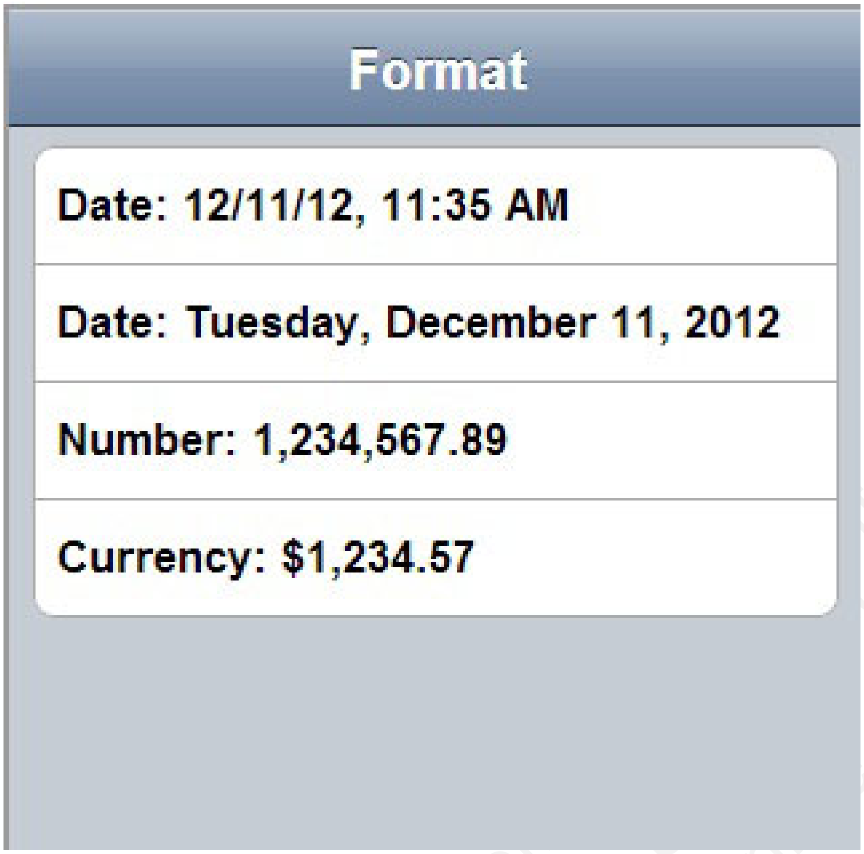
Day

Format

Icon

Locale: en-US

Figure 87. Dojo globalization application



Format

Date: 12/11/12, 11:35 AM

Date: Tuesday, December 11, 2012

Number: 1,234,567.89

Currency: $1,234.57

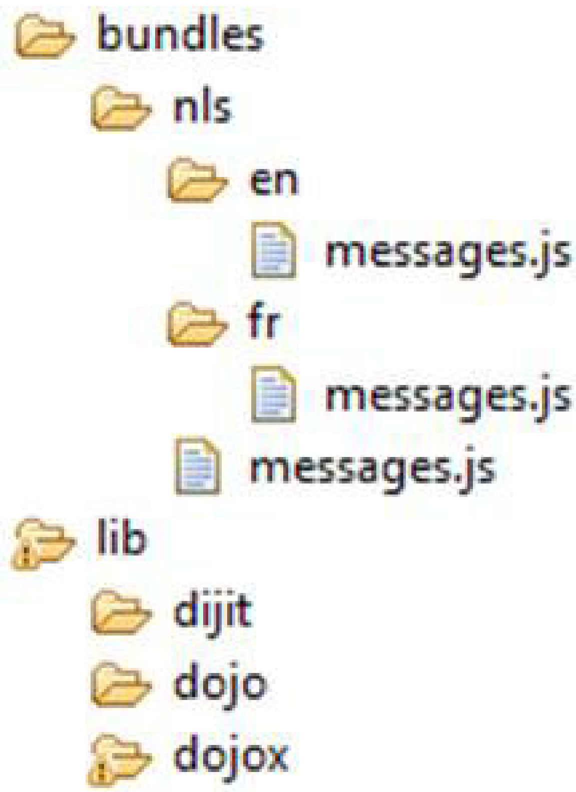
Figure 88. Dojo cultural formatting

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.



```
bundles
    nls
        en
            messages.js
        fr
            messages.js
        messages.js
lib
    dijit
    dojo
    dojox
```

Figure 89. 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"></h
  <ul data-dojo-type="dojox.mobile.RoundRectList">
    <li id="months_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="months" callback="getMonths
    <li id="days_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="days" callback="getDays" labe
    <li id="formats_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="formats" callback="getFor
    <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_foote
</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_icon"
</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_mon
</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.

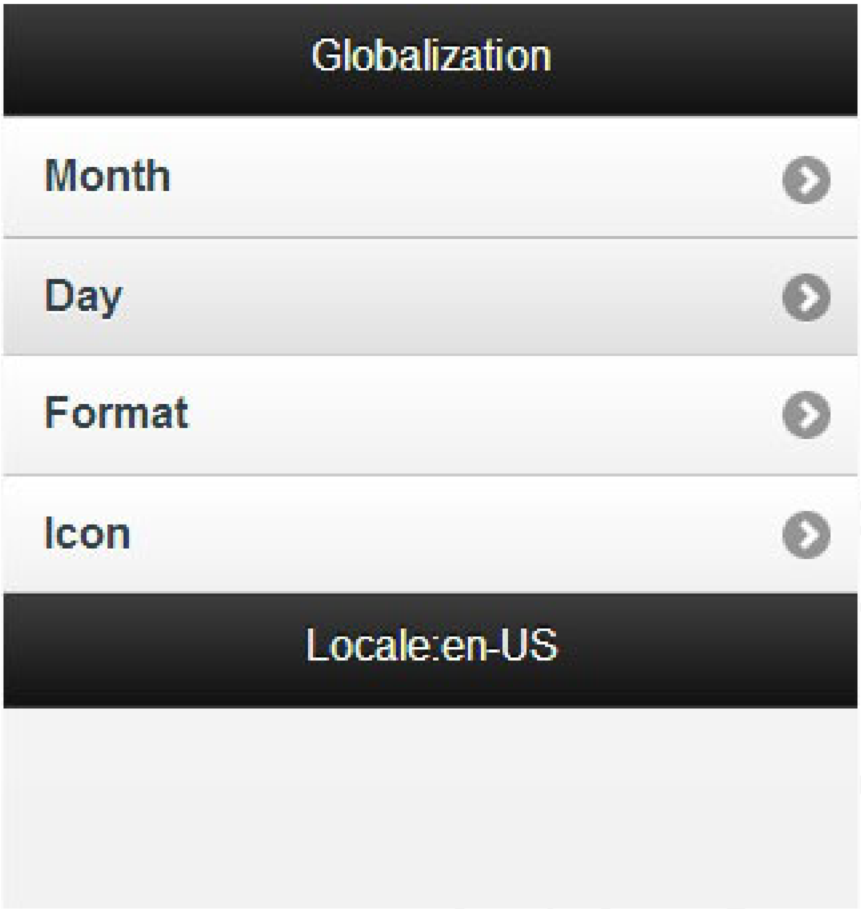


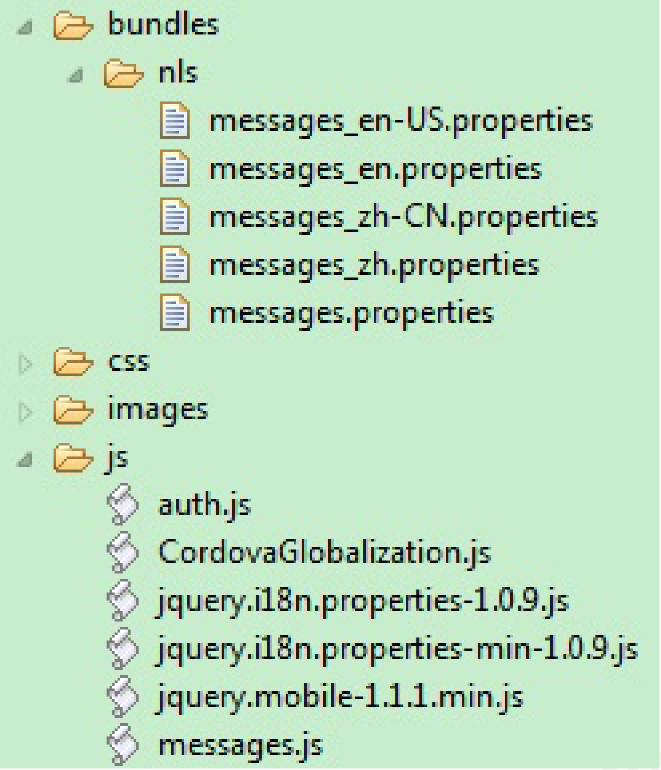
Figure 90. jQuery globalization application

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.



```
bundles
  nls
    messages_en-US.properties
    messages_en.properties
    messages_zh-CN.properties
    messages_zh.properties
    messages.properties
css
images
js
  auth.js
  CordovaGlobalization.js
  jquery.i18n.properties-1.0.9.js
  jquery.i18n.properties-min-1.0.9.js
  jquery.mobile-1.1.1.min.js
  messages.js
```

Figure 91. 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.

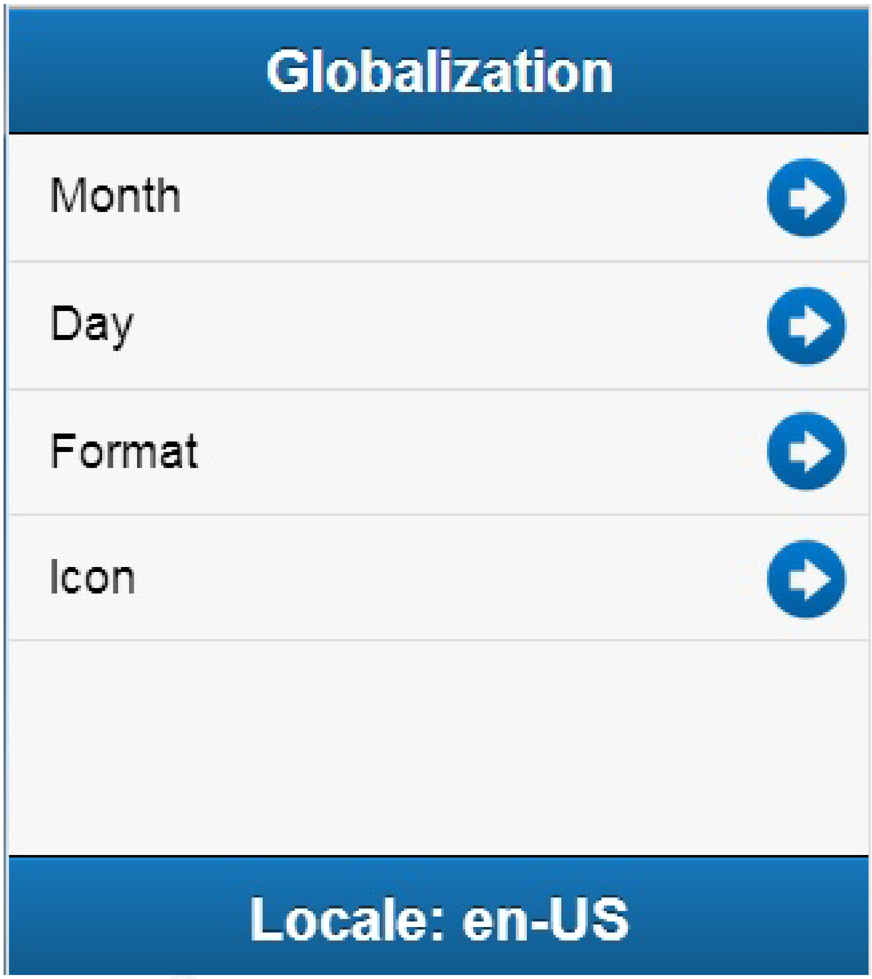


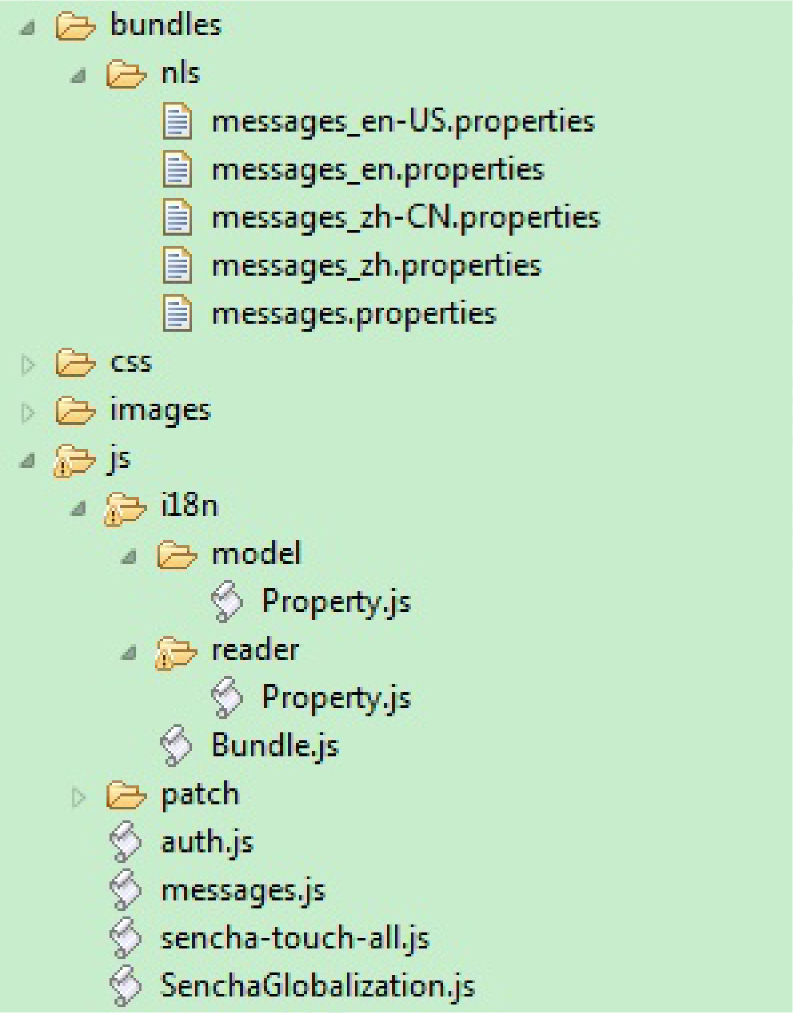
Figure 92. Sencha Touch globalization application

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.



```
bundles
    nls
        messages_en-US.properties
        messages_en.properties
        messages_zh-CN.properties
        messages_zh.properties
        messages.properties
css
images
js
    i18n
        model
            Property.js
        reader
            Property.js
        Bundle.js
    patch
    auth.js
    messages.js
    sencha-touch-all.js
    SenchaGlobalization.js
```

Figure 93. 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 102. 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.

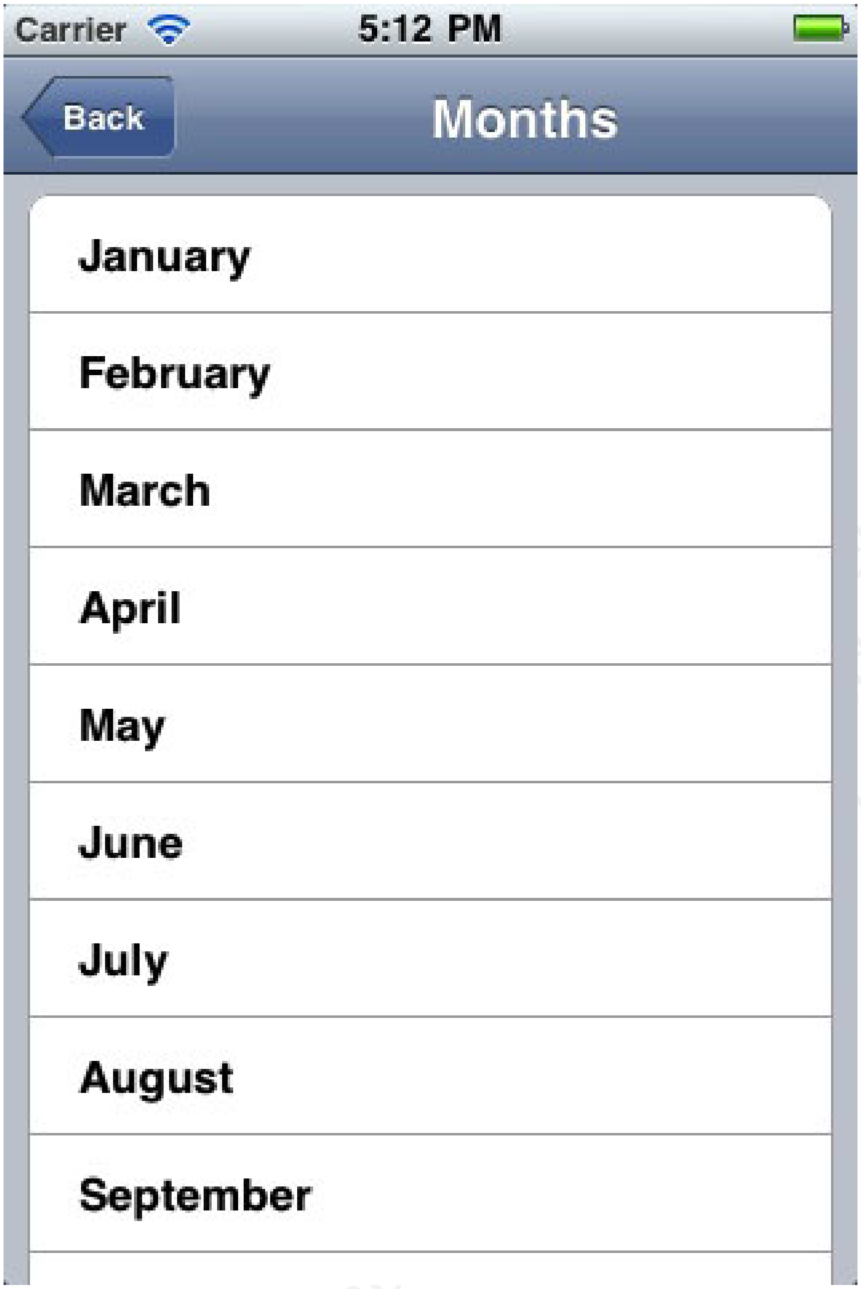
Table 103. Listing 1: Using the Cordova globalization API

```
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 104. 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);
  });
};
```



January
February
March
April
May
June
July
August
September

Figure 94. Using Cordova to show locale-based months

Table 105. 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"}
  );
}
```

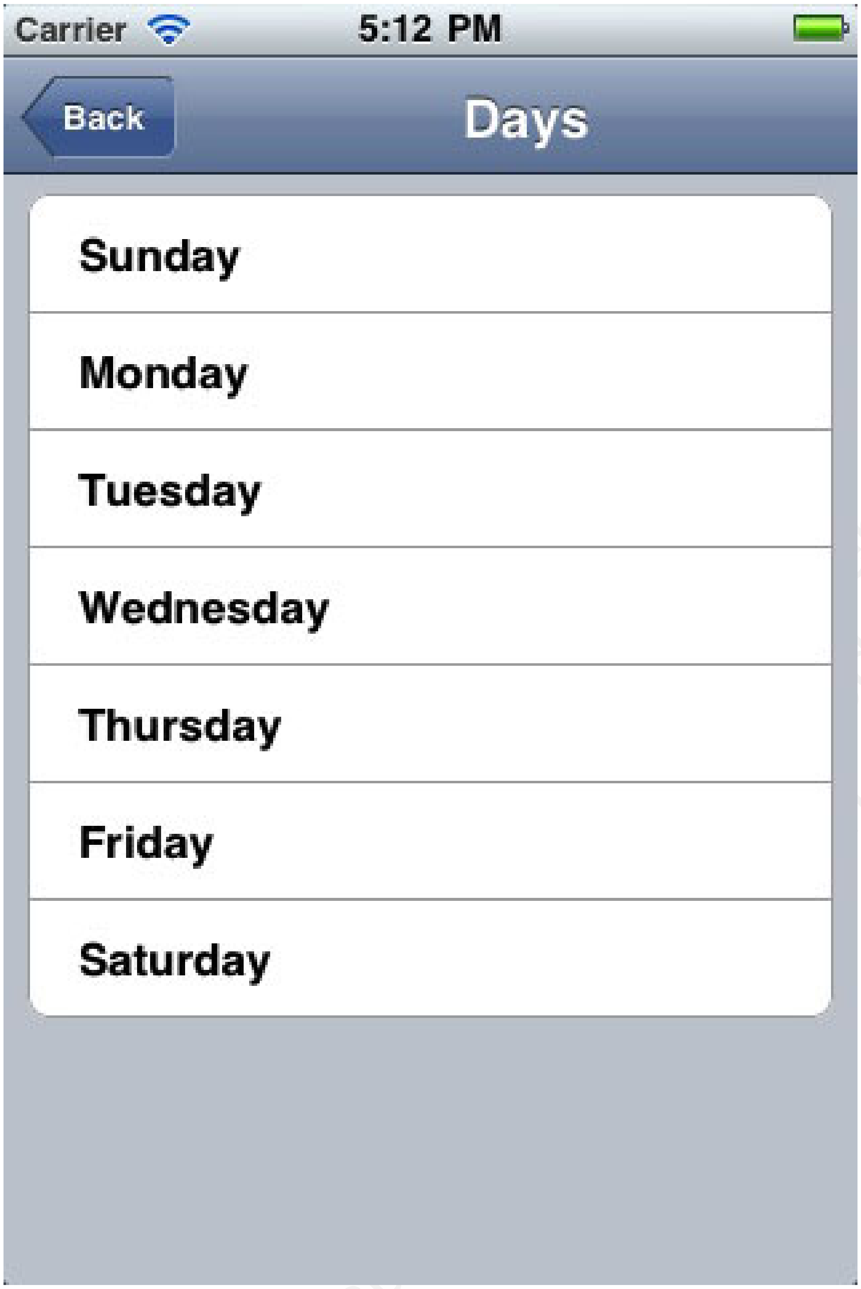
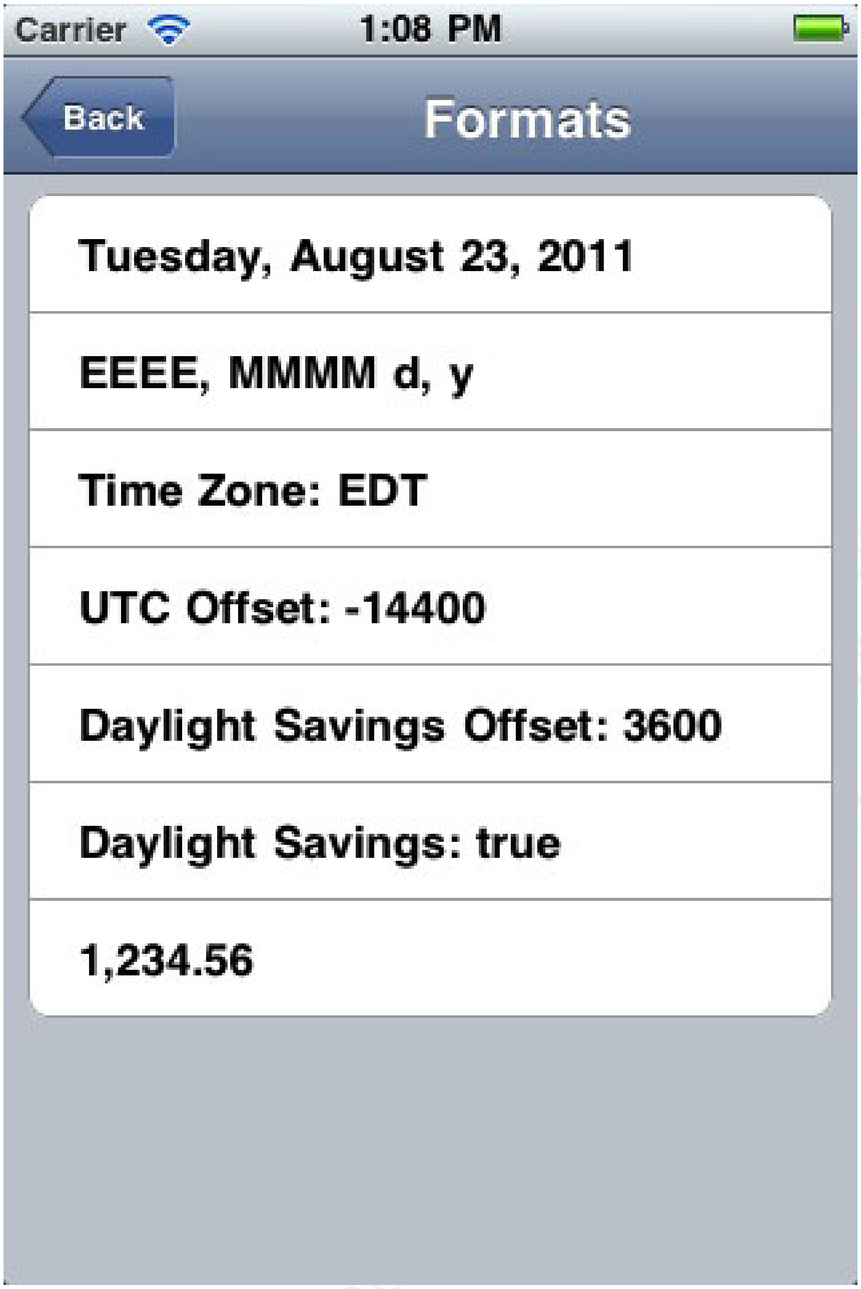


Figure 95. Using Cordova to show the days of week

Table 106. 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"}
  );
}
```



Tuesday, August 23, 2011

EEEE, MMMM d, y

Time Zone: EDT

UTC Offset: -14400

Daylight Savings Offset: 3600

Daylight Savings: true

1,234.56

Figure 96. 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.

Enabling translation of system messages

You can enable the translation of the system messages that the application displays, such as Internet connection is not available, or Invalid user name or password.

You can find the list of the system messages in the worklight/messages/messages.json file that is in the environment folder of the projects that you generated with IBM Worklight.

To enable the translation of a system message, you must override its value in the WL.ClientMessages object, as indicated in "The WL.ClientMessages object" on page 699.

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 example:

   ```
   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 examples 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 = "□□□□□□□";
     Messages.actionsLabel = "□□□o□ □□□□a:";
     Messages.sampleText = "□□□ □□□□□□ □□□□□□ □□ □□□□□□□ □□□□□.";
   }
   ```

```
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);
}
```

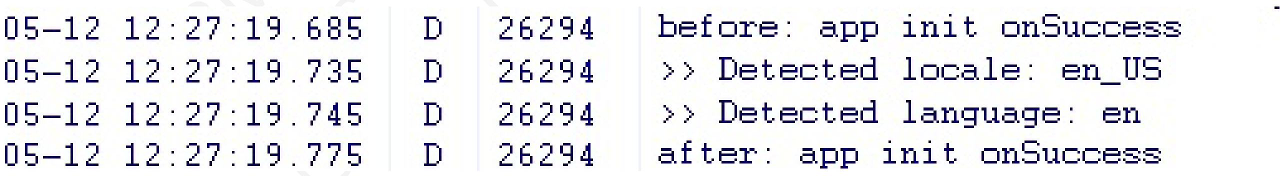
A language parameter is passed to the languageChanged() JavaScript function. The languageChanged() function calls the corresponding function to override the default English language string.

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.

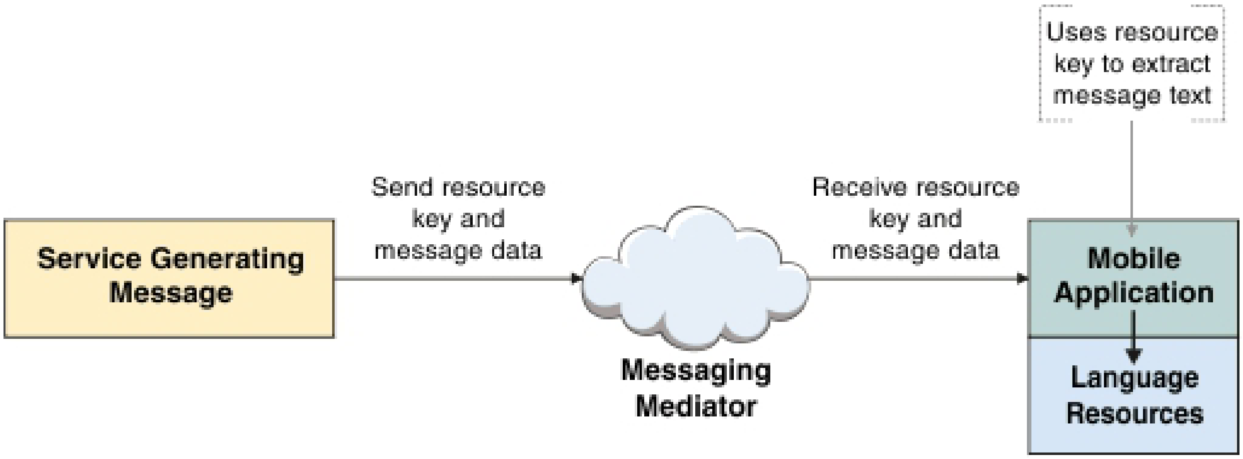

Uses resource key to extract message text
Send resource key and message data
Receive resource key and message data
Service Generating Message
Messaging Mediator
Mobile Application
Language Resources

Figure 97. 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 27

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.

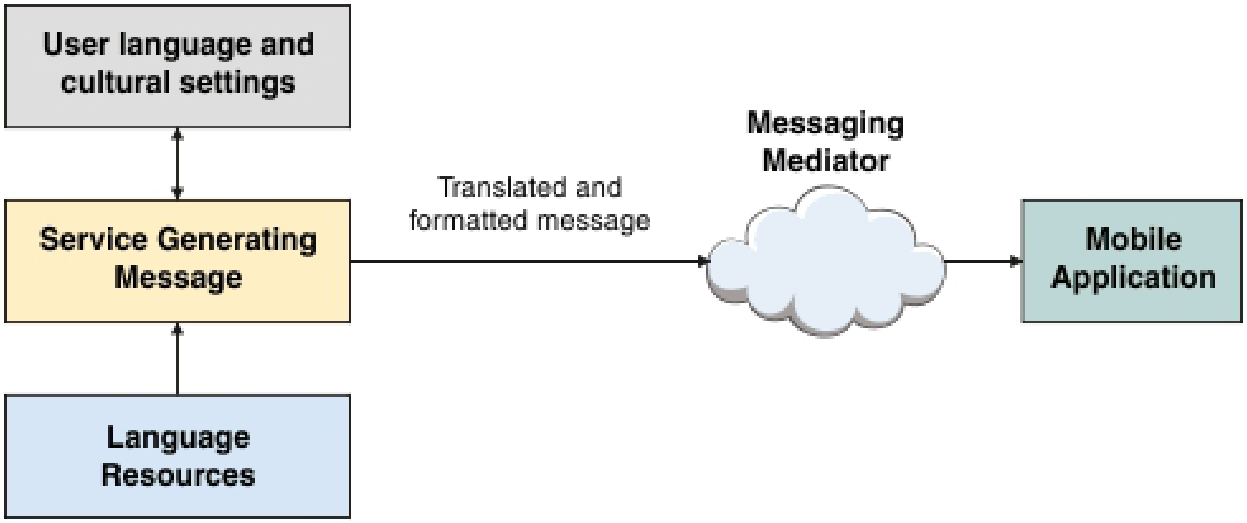

User language and cultural settings
Service Generating Message
Language Resources
Translated and formatted message
Messaging Mediator
Mobile Application

Figure 98. 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 in the following links. 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. IBM 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, iOS, and Windows Phone 8.

The following figure shows how the location services feature works:

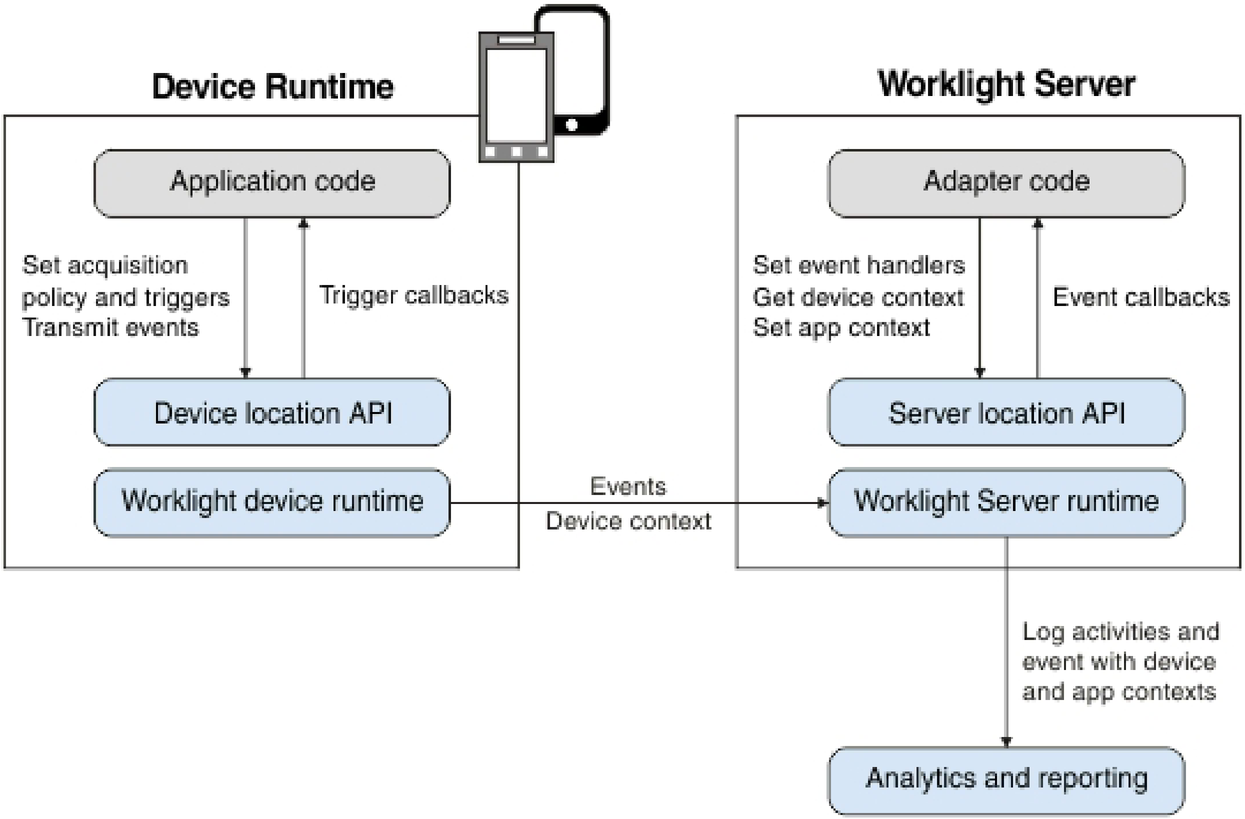


Figure 99. Location services architecture

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 synchronized 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, Android, and Windows Phone 8 in IBM Worklight V6.1.0. 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, Android, and Windows Phone 8:

Feature	iOS	Android	Windows Phone 8
Geo acquisition policy	minChangeTime is not supported.	highAccuracyOptions: iOSBestAccuracy 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.		Not supported. Only the Connect and Disconnect triggers are supported for Windows Phone 8 WiFi.
Connected WiFi signal strength	Not supported.		Not supported.
Connected WiFi MAC address			Not supported.
KeepAliveInBackground	Not supported. Use standard iOS options for acquiring location data while in the background.		Not supported. Use standard Windows Phone 8 options for acquiring location data while in the background.

For information about iOS, Android, and Windows Phone 8 permissions, see "Location services permissions."

Location services permissions

To use IBM Worklight location services, you must define the correct permissions.

Location services are supported for hybrid applications on Android, iOS, and Windows Phone 8.

Android

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

iOS

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)
```

Windows Phone 8

For Windows Phone 8, you must add the ID_CAP_LOCATION capability in the WMAppManifest.xml file.

When location services are running in the background on Windows Phone 8, replace the DefaultTask details in the WMAppManifest.xml file with the following information:

```
<DefaultTask Name="_default" NavigationPage="MainPage.xml"> <BackgroundExecution> <ExecutionType Name
```

See the Windows Phone Development Center web page http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662935(v=vs.105).aspx for details on running location-tracking apps in the background.

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 time period.
DwellOutside	The trigger is activated when the device remains outside a region for a given time period.

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 time period.
DwellOutside	The trigger is activated when the device remains outside a WiFi area for a given time period.
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.

You can control trigger activation based on confidence levels. A `low` confidence level is used to indicate that the WiFi acquisition policy **`signalStrengthThreshold`** value is used when determining whether an access point is visible. A `medium` confidence level is used to indicate that a signal strength of at least 50% is necessary for an access point to be visible. A `high` confidence level is used to indicate that a signal strength of at least 80% is necessary for an access point to be visible.

When you use the confidence level to determine whether an access point is visible, each specified access point in the area must be at least as strong as that indicated by the confidence level. If the area access point is for an SSID without a MAC address, then the highest signal strength for that SSID must be at least as strong as that indicated by the confidence level. In order to exit the area, the signal strength level for at least one access point must be below the WiFi acquisition policy **`signalStrengthThreshold`** value.

Note: For WiFi triggers, the **`confidenceLevel`** parameter is not supported by DwellOutside.

For detailed information about the parameters for the trigger types, see the `startAcquisition` method as defined in the WL.Device class.

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. Use of power 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 permissions” on page 661. 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.

For hybrid Android, iOS, or Windows Phone 8:

```
window.addEventListener("batterylow", goToPowerSaveMode, false);

function goToPowerSaveMode() {
  WL.Device.startAcquisition(
    { Geo: WL.Device.Geo.Profiles.PowerSaving() },
    triggers,
    failureFunctions
  );
}
```

For native Android:

```
WLDevice wlDevice = WLClient.getInstance().getWLDevice();
wlDevice.startAcquisition(
  new WLLocationServicesConfiguration()
    .setPolicy(new WLAcquisitionPolicy()
      .setGeoPolicy(WLGeoAcquisitionPolicy.getPowerSavingProfile()))
    .setTriggers(triggers)
      .setFailureCallbacks(failureFunctions)
  );
```

For native iOS:

```
id<WLDevice> wlDevice = [[WLClient sharedInstance] getWLDevice];
[ wlDevice startAcquisition:
  [
    [
      [
        [[WLLocationServicesConfiguration alloc] init]
        setPolicy:
        [
          [[WLAcquisitionPolicy alloc] init]
          setGeoPolicy: [WLGeoAcquisitionPolicy getPowerSavingProfile]
        ]
      ]
      setTriggers: triggers
    ]
    setFailureCallbacks: 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 664.

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

For hybrid Android, iOS, or Windows Phone 8:

```
function wlCommonInit(){

  /*
   * Application is started in offline mode as defined by a connectOnStartup property in initOptio
   * 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
   *    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
   *    Don't forget to specify and implement onSuccess and onFailure callback functions for WL.Cl
   *
   *    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
      }
    }
  }
};
```

For native Android:

```
WLTriggersConfiguration triggers = new WLTriggersConfiguration();
triggers.getGeoTriggers().put("centralPark",
   new WLGeoDwellInsideTrigger()
     .setArea(new WLPolygon(Arrays.asList(
            new WLCoordinate(40.80062106285157, -73.95824432373092),
            new WLCoordinate(40.79691751000037, -73.94948959350631),
            new WLCoordinate(40.764486356929645, -73.97309303283704),
            new WLCoordinate(40.76799670467469, -73.98167610168441))))
     .setDwellingTime(600000) // 10 minutes
     .setBufferZoneWidth(-100) // at least 100 meters within the park
     .setCallback(after10MinsInCentralPark));
triggers.getGeoTriggers().put("statueOfLiberty",
   new WLGeoEnterTrigger()
     .setArea(new WLCircle(new WLCoordinate(40.689167, -74.044444), 5000)) // 5 km
     .setConfidenceLevel(WLConfidenceLevel.HIGH) // ⌂95% confidence that we are in the circle
     .setEvent(new JSONObject("{nearAttraction: 'statue_of_liberty'}"))
     .setTransmitImmediately(true));
```

For native iOS:

```
WLTriggersConfiguration* triggers = [[WLTriggersConfiguration alloc] init];

[[triggers getGeoTriggers] setObject:
   [[[[
      [[WLGeoDwellInsideTrigger alloc] init]
         setArea: [[WLPolygon alloc] init: [NSMutableArray arrayWithObjects:
                  [[WLCoordinate alloc] initWithLatitude:40.80062106285157 longitude:-73.95824432373
                  [[WLCoordinate alloc] initWithLatitude:40.79691751000037 longitude:-73.94948959350
                  [[WLCoordinate alloc] initWithLatitude:40.764486356929645 longitude:-73.97309303283
                  [[WLCoordinate alloc] initWithLatitude:40.76799670467469 longitude:-73.98167610168
                  nil]]]
         setDwellingTime: 60000] // 10 minutes
         setBufferZoneWidth: -100] // at least 100 meters within the park
         setCallback: after10MinsInCentralPark]
   forKey:@"centralPark"];


[[triggers getGeoTriggers] setObject:
   [[[[
      [[WLGeoEnterTrigger alloc] init]
         setArea: [[WLCircle alloc] initWithCenter:[[WLCoordinate alloc] initWithLatitude:40.689167
         setConfidenceLevel: HIGH] // ⌂95% confidence that we are in the circle
         setEvent: [NSDictionary dictionaryWithObject: @"statue_of_liberty" forKey:@"nearAttraction"
         setTransmitImmediately: true]
   forKey:@"statueOfLiberty"];
```

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 required 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 - hybrid Android, iOS, and Windows Phone 8

```
function wlCommonInit(){

  /*
   * Application is started in offline mode as defined by a connectOnStartup property in initOptic
   * 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
   *    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
   *    You must specify and implement onSuccess and onFailure callback functions for WL.Client.co
   *
   *    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
      }
    }
  );
}
```

Application to set up acquisition policy, including triggers - native Android

```
Set<String> ssids = new HashSet<String>();

public void addNetworkToBeScanned(String ssid) {
    ssids.add(ssid);
}

public void removeNetwork(String ssid) {
  ssids.remove(ssid);
}

public void startScanning() throws JSONException {
  List<WLWifiAccessPointFilter> filters = new ArrayList<WLWifiAccessPointFilter>();

   for (String ssid : ssids)
     filters.add(new WLWifiAccessPointFilter (ssid, WLWifiAccessPointFilter.WILDCARD));

  WLAcquisitionPolicy policy = new WLAcquisitionPolicy()
    .setWifiPolicy(
       new WLWifiAcquisitionPolicy()
         .setInterval(3000)
         .setAccessPointFilters(filters));

  WLTriggersConfiguration triggers = new WLTriggersConfiguration();
  triggers.getWifiTriggers().put(
       "change",
       new WLWifiVisibleAccessPointsChangeTrigger()
         .setEvent(new JSONObject("{name: 'moved'}")));

  WLAcquisitionFailureCallbacksConfiguration failures = new WLAcquisitionFailureCallbacksConfigura
  failures.setWifiFailureCallback(new WLWifiFailureCallback() {
     @Override
     public void execute(WLWifiError wifiError) {
        onWifiFailure(wifiError);
     }
  });

  WLClient.getInstance().getWLDevice().startAcquisition(new WLLocationServicesConfiguration()
    .setPolicy(policy)
    .setTriggers(triggers)
    .setFailureCallbacks(Collections.singletonList(failures)));
  }

  void stopScanning() {
    WLClient.getInstance().getWLDevice().stopAcquisition();
  }

  void onWifiFailure(WLWifiError wifiError) {
    //show an error message to the user...
  }

  // receives a string, indicating the name of the region
  void setCurrentRegion(String region) {
     WLProcedureInvocationData invocData = new WLProcedureInvocationData ("HT_WifiScan", "setAppCo
     invocData.setParameters(new Object[] {"{regionName: '" + region + "'}"});
```

```
 WLClient.getInstance().invokeProcedure(
  invocData,
  new WLResponseListener() {

   @Override
   public void onSuccess(WLResponse response) {
    // update UI, indicating success
   }

   @Override
   public void onFailure(WLFailResponse response) {
    // update UI, indicating error
   }
  });
}
```

Application to set up acquisition policy, including triggers - native iOS

```
// NSMutableSet* ssids  -- is defined in the header as a member field and initialized as ssids = [NSM

 -(void) addNetworkToBeScanned: (NSString*) ssid {
     [ssids addObject:ssid];
}

 -(void) removeNetwork: (NSString*) ssid {
     [ssids removeObject:ssid];
 }

-(void) startScanning {
     NSMutableArray* filters = [[NSMutableArray alloc] init];

    for (NSString* ssid in ssids) {
      [filters addObject: [[WLWifiAccessPointFilter alloc] initWithSSID:ssid MAC:WILDCARD]];
   }

    WLAcquisitionPolicy* policy =
     [
      [[WLAcquisitionPolicy alloc] init]
        setWifiPolicy:
         [[
          [[WLWifiAcquisitionPolicy alloc] init]
            setInterval: 3000]
            setAccessPointFilters: filters]];

    WLTriggersConfiguration* triggers = [[WLTriggersConfiguration alloc] init];
    [[triggers getWifiTriggers] setObject:
     [
      [[WLWifiVisibleAccessPointsChangeTrigger alloc] init]
        setEvent: [NSDictionary dictionaryWithObject: @"moved" forKey:@"name"]]
     forKey:@"change"];

 WLAcquisitionFailureCallbacksConfiguration* failures = [[WLAcquisitionFailureCallbacksConfiguratio
 [failures setWifiFailureCallback: [WLCallbackFactory createWifiFailureCallback:^(WLWifiError* wifiE
   [self onWifiError: wifiError];
     }]];

 [[[WLClient sharedInstance] getWLDevice] startAcqusition:
  [[[
    [[WLLocationServicesConfiguration alloc] init]
     setPolicy: policy]
     setTriggers: triggers]
     setFailureCallbacks: [NSMutableArray arrayWithObject:failures]]];
}

 -(void) stopScanning {
  [[[WLClient sharedInstance] getWLDevice] stopAcqusition];
 }
```

```
  -(void) onWifiFailure: (WLWifiError*) wifiError {
    //show an error message to the user...
  }

  // receives a string, indicating the name of the region
  -(void) setCurrentRegion: (NSString*) region {
    WLProcedureInvocationData* invocData = [[WLProcedureInvocationData alloc] initWithAdapter:@"HT
    [invocData setParameters:[NSArray arrayWithObject:[NSString stringWithFormat:@"{regionName: '%

    // replace this with a WLDelegate instance which will update the UI indicating success/failure
    id<WLDelegate> delegate = nil;

  {[WLClient sharedInstance] invokeProcedure:invocData withDelegate: delegate];
  }
```

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 - hybrid Android, iOS, and Windows Phone 8

```
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
     }
   }
};
```

Example of Enter trigger - native Android

```
WLTriggersConfiguration triggers = new WLTriggersConfiguration();

triggers.getWifiTriggers().put(
     "welcomeToMall",
     new WLWifiEnterTrigger()
       .setAreaAccessPoints(Collections.singletonList(new WLWifiAccessPointFilter("FreeMallWifi"))
       .SetCallback(showWelcome));

triggers.getWifiTriggers().put(
     "foodCourt",
     new WLWifiEnterTrigger()
        .setAreaAccessPoints(Arrays.asList(
           new WLWifiAccessPointFilter("FreeMallWifi", "12:34:56:78:9A:BC"),
           new WLWifiAccessPointFilter("FreeMallWifi", "CB:A9:87:65:43:21")))
        .SetCallback(showFoodCoupons));
```

Example of Enter trigger - native iOS

```
 WLTriggersConfiguration triggers = new WLTriggersConfiguration();

WLTriggersConfiguration* triggers = [[WLTriggersConfiguration alloc] init];
[[triggers getWifiTriggers] setObject:
   [[
     [[WLWifiEnterTrigger alloc] init]
       setAreaAccessPoints: [NSMutableArray arrayWithObject: [[WLWifiAccessPointFilter alloc] init:
       setCallback: showWelcome]
   forKey:@"welcomeToMall"];

[[triggers getWifiTriggers] setObject:
   [[
     [[WLWifiEnterTrigger alloc] init]
       setAreaAccessPoints: [NSMutableArray arrayWithObjects:
          [[WLWifiAccessPointFilter alloc] initWithSSID: @"FreeMallWifi" MAC: @"12:34:56:78:9A:BC"],
          [[WLWifiAccessPointFilter alloc] initWithSSID: @"FreeMallWifi" MAC: @"CB:A9:87:65:43:21"],
          nil]]
       setCallback: showFoodCoupons]
   forKey:@"foodCourt"];
```

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*60000) { // 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 -hybrid Android, iOS, and Windows Phone 8

```
function wlCommonInit(){

  // 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);
}
```

Mobile application logic - native Android

```
// get truck id (for example from the user) -- for this example, using a hard-coded value.
long truckId = 123;
String driverName = "John Smith";

WLAcquisitionPolicy policy = new WLAcquisitionPolicy()
  .setGeoPolicy(new WLGeoAcquisitionPolicy()
     .setEnableHighAccuracy(true)
     .setTimeout(10000));

WLTriggersConfiguration triggers = new WLTriggersConfiguration();
triggers.getGeoTriggers().put(
  "tracking",
  new WLGeoPositionChangeTrigger()
    .setMinChangeDistance(100)
    .setEvent(new JSONObject()
          .put("vehicle", "DeliveryTruck")
          .put("id", truckId)
          .put("driverName", driverName)));
```

```
WLClient.getInstance().getWLDevice().startAcquisition
  new WLLocationServicesConfiguration()
    .setPolicy(policy)
    .setTriggers(triggers));
```

Mobile application logic - native iOS

```
// get truck id (for example from the user) -- for this example, using a hard-coded value.
long long truckId = 123;
NSString* driverName = @"John Smith";

WLAcquisitionPolicy* policy =
  [[WLAcquisitionPolicy alloc] init]
    [
      setGeoPolicy:
        [[
           [[WLGeoAcquisitionPolicy alloc] init]
           setEnableHighAccuracy: true]
           setTimeout: 10000]];

WLTriggersConfiguration* triggers = [[WLTriggersConfiguration alloc] init];
[[triggers getGeoTriggers] setObject:
  [[
    [[WLGeoPositionChangeTrigger alloc] init]
        setMinChangeDistance: 100]
        setEvent:  [NSDictionary dictionaryWithObjectsAndKeys:
           @"DeliveryTruck", @"vehicle",
           truckId, @"id",
           driverName, @"driverName",
           nil]]
   forKey:@"tracking"];

[[[WLClient sharedInstance] getWLDevice] startAcqusition:
  [[
    [[WLLocationServicesConfiguration alloc] init]
        setPolicy: policy]
        setTriggers: 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, iOS, or Windows Phone 8, 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 and hybrid applications, 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 the method `setKeepAliveInBackground` as defined in the WL.App class.

- For Android devices and native applications, you should access the location APIs through a service. For more information about Android services, see the “Services” section on the Android development site at http://developer.android.com/guide/components/services.html.
- 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.
- For Windows Phone 8 devices, replace the DefaultTask details in the `WMAppManifest.xml` file with: `<DefaultTask Name="_default" NavigationPage="MainPage.xml"> <BackgroundExecution> <ExecutionType Name="LocationTracking" /> <BackgroundExecution> </DefaultTask>`. See the Windows Phone Development Center web page http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj662935(v=vs.105).aspx for details on running location-tracking apps in the background.

Testing hybrid location service applications with MBS

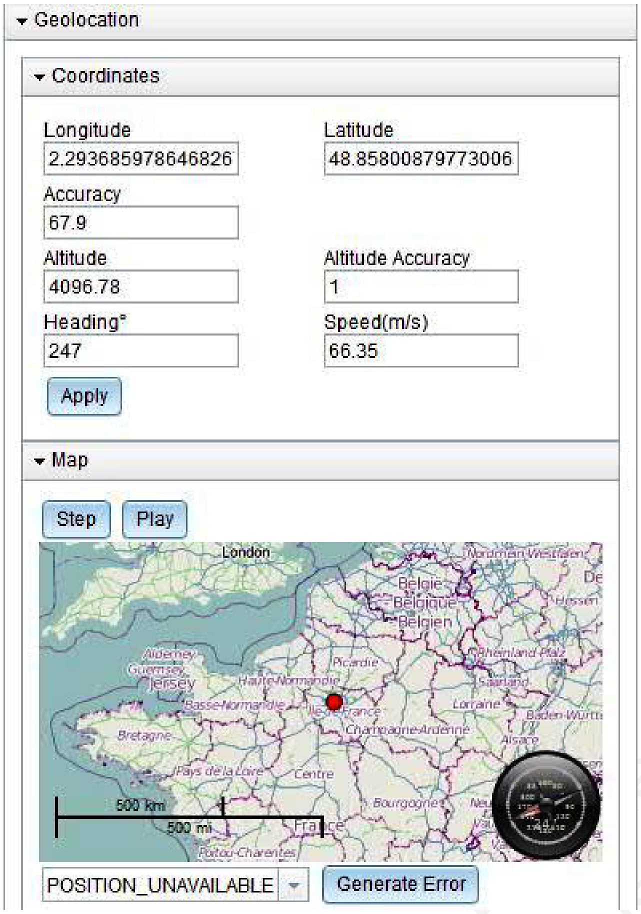
You can use the mobile browser simulator (MBS) to test applications within a browser, and preview Worklight applications on Android, iOS, and Windows Phone 8. Location services only support these platforms, other platforms must be removed. With the Geolocation, Network and Scenario widgets, you can test applications in MBS that use the JavaScript location service APIs.

Mobile browser simulator geolocation widget

The geolocation widget can be used to provide a simulation of the device's geolocation information to the application. The application can access this information by using the W3C geolocation APIs or IBM Worklight location services APIs for hybrid applications.

Note: Location services for hybrid applications are only supported for Android, iOS, and Windows Phone 8. Other platforms must be removed.

The geolocation information can be directly configured in the widget.



Geolocation

Coordinates

Longitude: 2.293685978646826

Latitude: 48.85800879773006

Accuracy: 67.9

Altitude: 4096.78

Altitude Accuracy: 1

Heading°: 247

Speed(m/s): 66.35

You can use the **Latitude** and **Longitude** options to set specific GPS coordinates. You can click the map to update the latitude and longitude. A **Heading** of 0º corresponds to North. By clicking the **Play**, the device's movement is simulated from the current location in the direction that is given by **Heading**, at the speed specified. An update is given once a second. After you click **Play**, the button changes to **Stop**, and clicking it stops the simulation. Alternatively, a single 1-second step can be taken by using **Step**.

To simulate various errors that might occur, select the appropriate error and click **Generate Error**. This action causes the next call to a geolocation API to have its failure function is called with the selected error.

Accuracy is used to set the accuracy of the position, and can affect geofences when you are using the `confidenceLevel` parameter. For more information, see “Triggers” on page 662.

Altitude and **Altitude Accuracy** appear in the position information, but are not used by location services APIs.

You can use **Step** and **Play** to simulate the movement of devices. For example, if the **Speed** setting is increased, you can see the effect in the simulator window by clicking **Step** to generate a new set of values for the app, or **Play** to generate new values periodically.

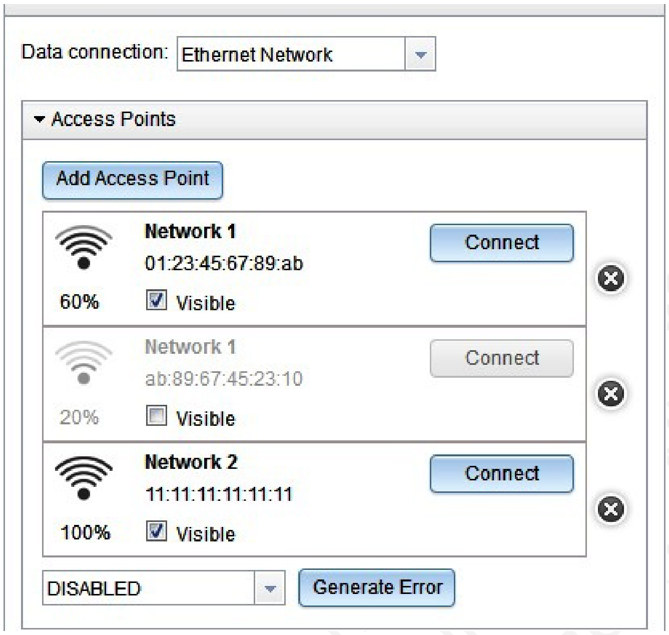
Mobile browser simulator network widget

The network widget can be used to provide network information, for example information that is accessible by the `WL.Device.getNetworkInfo` API or that can activate WiFi based triggers in IBM Worklight location services.

WiFi access points can be configured in this widget for testing the use of location services, for example see the section on WiFi triggers in “Triggers” on page 662.

Click **Add Access Point** to define a new access point. There you must define the **SSID** and **MAC** addresses. You can also specify visibility and signal strength.

Click an access point to open a dialog to edit its properties.



Data connection: Ethernet Network

Network 1
01:23:45:67:89:ab
60% ☑ Visible

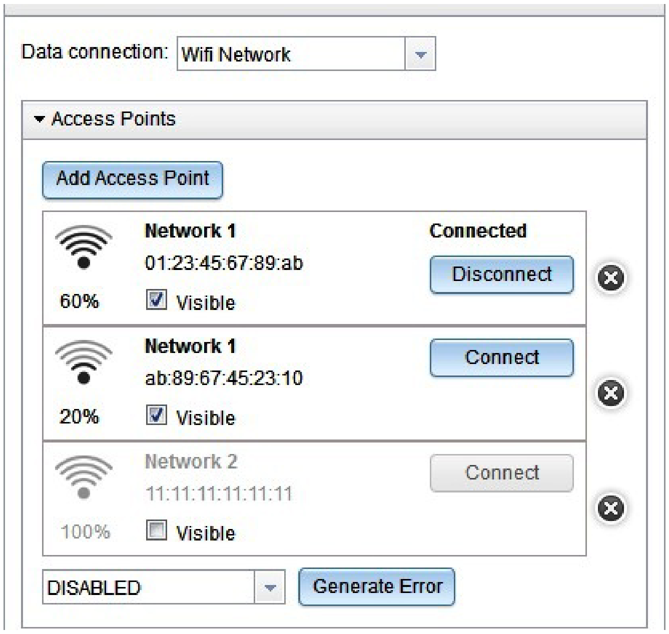
Network 1
ab:89:67:45:23:10
20% □ Visible

Network 2
11:11:11:11:11:11
100% ☑ Visible

DISABLED

The **Visible** check box indicates whether the access point is visible to the device, and whether it could be returned by a call to `WL.Device.Wifi.acquireVisibleAccessPoints`.

Click **Connect** to set an access point to be the connected access point, and **Disconnect** to disconnect it. Clicking **Connect** also changes the data connection to WiFi Network.



Data connection: Wifi Network

Only one access point can be connected at a time. The access point is connected to the data network, which switches to WiFi. When you switch the network to something that is not WiFi, then the connected access point is disconnected.

To simulate various errors that can occur, select the appropriate error from the drop-down list and click **Generate Error**. This action causes the next acquisition that is performed to call its failure function with the selected error.

Location scenarios

Location scenarios are intended for developers who want to develop and test location-dependent behavior of mobile apps, as part of the IBM Worklight mobile tools suite. Location scenarios represent a multi-sensor simulation environment for mobile apps.

Note: The scenario widget only runs on the following web browsers:
- Chrome 17 and later.
- Safari 5 and later.

Worklight Mobile Browser Simulator (MBS) allows developers to quickly test, debug, and experiment with their hybrid applications from the convenience of the web browser. Although MBS provides sensor simulation it requires manual updates each time a new sensor data update is required, which is slow and not repeatable. With the new location scenario widget, testing location-aware apps becomes much easier.

The process for working with a location scenario widget is as follows:

1. The user defines a scenario on a map, which comprises the following items:
   - A route, with the time for each point
   - WiFi hot spots
   - No-GPS zones
2. As the user plays back the scenario, the sensor inputs are simulated:
   - Location updates arrive according to the route and no-GPS zone definitions. These include the generation of timeout errors based on the acquisition policy, when the scenario moves through no-GPS zones.

- WiFi hot spots become visible according to the current location. These hot spots include the calculated signal strength on each update.

The testing scenarios that are listed here demonstrate the versatility of the scenario widget.

Background: You have implemented an application for improved hotel check-in. This application defines a geo-fence with a five-kilometer radius around a hotel, within which a guest is invited to check in. In addition, the application defines a WiFi trigger when the hotel WiFi is visible, to welcome the hotel guest and alert the hotel manager that the guest has arrived.

Testing a scenario for entering a geofence: You would like to test the geofence that is defined in this application. Complete the following steps:

1. Right-click the environment you want to test (hybrid Android, iOS, or Windows Phone 8) and click **Preview** to start the mobile browser simulator and open the scenario widget.
2. Create a new scenario and move to the correct location on the map. Create the path to the hotel by clicking the map to indicate the points on the path. For each path point you can specify the time. When completed, you can save the scenario to a local file.
3. Click **Play** . The current location on the route is clearly visualized for the application, as it is simulated. When the current location is five kilometers from the hotel (the defined trigger), you see a message appear on the mobile application indicating that the correct event was triggered and handled by your application, prompting the guest to check in remotely.

Testing a scenario for entering a WiFi zone: You would like to test your application when the guest enters the hotel. Complete the following steps:

1. Edit the scenario widget and add a visible network with the hotel SSID and MAC address in the hotel location.
2. Click **Play** . You can clearly see the current location that is simulated for your application. When the first geo-fence is entered, you work with the application to confirm the remote check-in. While you are operating the application the current location is still changing, as planned. As the current location enters the hotel WiFi zone, you can see that the correct event was triggered and handled by your application, welcoming the guest to the hotel.

Loading and editing a location scenario: You would like to test a scenario you created in the past. Complete the following steps:

1. Open the scenario widget and click **Import** to select a scenario.
2. Edit the scenario by moving some of the points in the route, by deleting some of the points in the route, and by adding more points to the route. When you have finished editing, click **Apply**. You can then play the scenario to test your application. To save the scenario to a file that you can play in the future, click **Export**.

Manually editing a location scenario: Complete the following steps:

1. Open a scenario file that you have tested in a text editor. The format is readable and you can clearly understand how the path, WiFi zones, and no-GPS zones are represented.
2. Manually edit the no-GPS polygon to a different polygon. You save the file and load it in the MBS. You can see the new polygon on the scenario map.

Location scenarios: demonstration:

This demonstration shows what is provided by the scenario widget, which can be used to develop and test location-dependent behavior of mobile apps, as part of the IBM Worklight mobile tools suite.

Clicking the scenario widget presents a scenario template. You can create, import, load, export, edit scenarios, and play or restart scenarios.

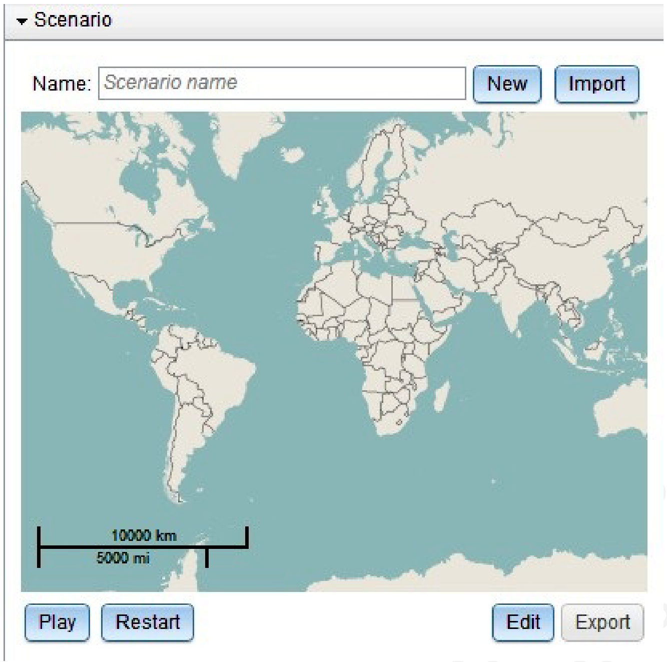


Figure 100.

- To name the scenario, enter a name in the **Name** field.
- To edit the scenario, click **Edit**.

Note: You provide the longitude and latitude for the location, which is based on the behavior of the application you want to test. To test geofences, enter a location close to their boundary. For example, if your application defines a geofence around a store, you could provide the location of that store.

Figure 2 shows a map displaying an area of roads and buildings and also includes the location of a shopping mall. The template supplies the **Longitude, Latitude** field which must be completed.

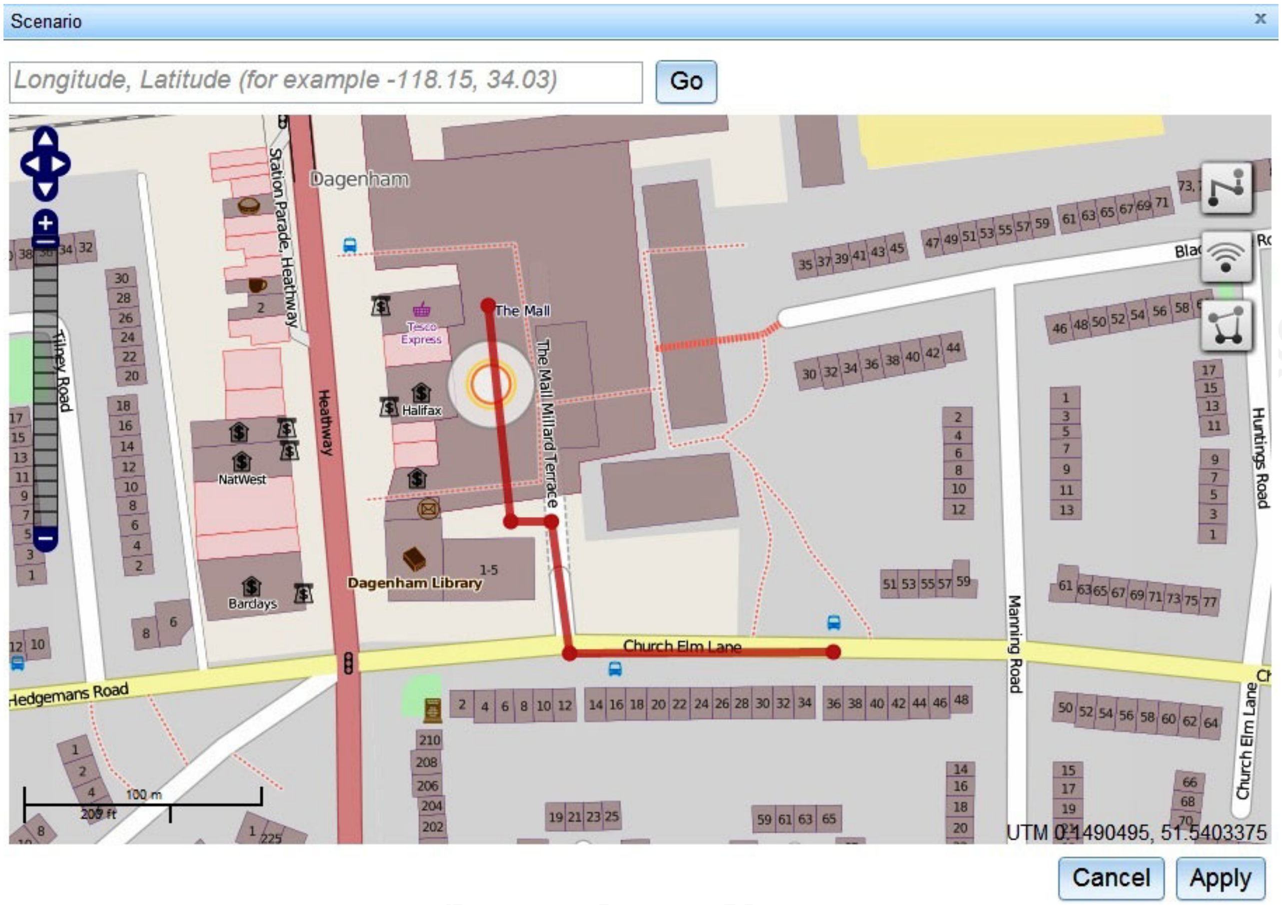


Figure 101.

Provide the location for the user's device to be set up and tested in the **Longitude, Latitude** field. This is where the movement of the device will be simulated. After entering the longitude and latitude click **Go** to move the map to that location.

The template also shows the various buttons and tools that are used for navigating, and for creating paths and zones in your scenario.

- To move around the map click the directional arrows or click and drag the map.
- To zoom in or out use the +/- track below the directional arrows, or use the scroll wheel.
- These buttons are provided for use, as necessary.
  - Click **Go** to move the map to the specified location.
  - Click **Cancel** to cancel all changes made to the scenario and close the editor.
  - Click **Apply** to apply the changes to the scenario and close the editor.

The three **icon** buttons are used to define the scenario by creating paths and zones.

- The first button is used to create a simulated path for the mobile subscriber. For example, you could simulate defining a route along a road, then turning into a parking lot, and then entering the shopping mall.
  - Click this button to define a path. Click the map to add vertices (points) to the path. Double-clicking adds the final point to the path.

  - Click a vertex to change the time at which the simulated subscriber will pass that vertex. The first vertex is fixed at time 0. When you click on a vertex, a **Delete Path** button is also visible, which deletes the entire path if clicked.
  - Click on the path between vertices to show two blue circles. The circle on the path can be clicked and dragged to move the path. The circle to the lower right of the path can be clicked and dragged to resize or rotate the path.
  - Only a single path can be defined at one time. Beginning a new path deletes the previously defined path.

  When you play the scenario, the geolocation widget automatically updates the location, the heading and the calculated speed, based on the simulated user's position along the path.

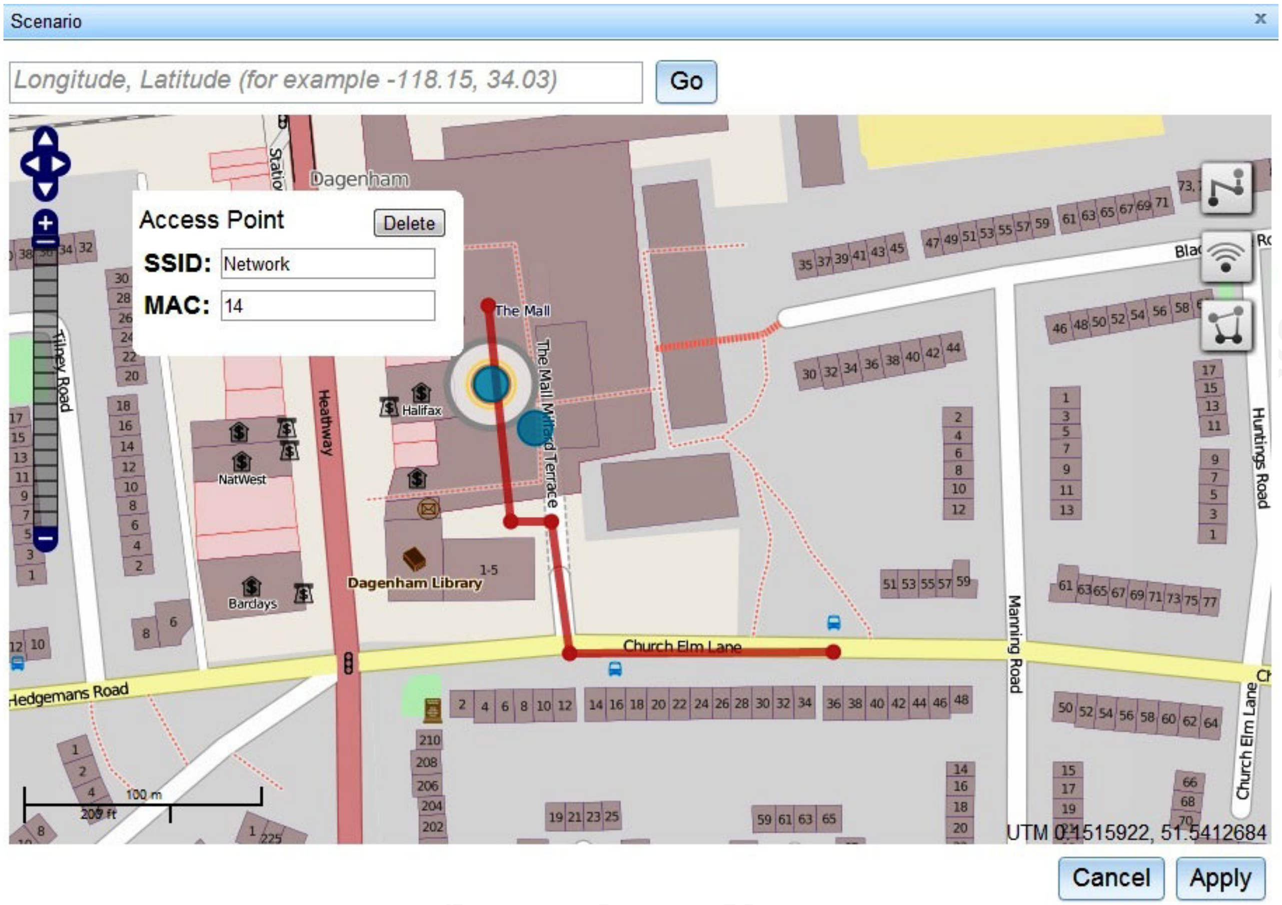
- The second button can be used to define WiFi access points and their coverage zones.
  - After you have clicked the button, clicking on the map defines a new access point. Click on the map where the center of the coverage zone for the WiFi access point should be, and drag to set the desired size. The coverage zone is displayed. The gray outer circle corresponds to a signal strength of 15%, which is the default signal strength threshold for the WiFi acquisition policy. The yellow inner circle corresponds to a 50% signal strength, which is the medium confidence level. The innermost orange circle indicates 80% signal strength, which is the high confidence level.
  - Click on the WiFi zone to edit the properties for the access point. You can set the SSID and MAC, or you can click **Delete** to delete the access point.
  - After clicking on a WiFi zone you can also see two blue circles. Clicking the circle in the center of the zone and dragging it can be used to move the WiFi zone. The circle to the lower left of the zone can be clicked and dragged to resize the WiFi zone.
  - It is possible to define multiple WiFi zones.

  When you play the scenario, the network widget automatically updates the visible access points, including their simulated signal strengths, based on the simulated device position. WiFi zones are not used to simulate the geolocation of the device.

- The third button is used to define no-GPS coverage zones.
  - Click this button to define no-GPS coverage zones. Click the map to add vertices (points) to define the boundary of the zone. Double-clicking adds the final point to the zone.
  - Click a no-GPS zone to move it, or delete it by clicking **Delete**. After you click a no-GPS zone, two blue circles appear. The blue circle at the center of the no-GPS zone can be clicked and dragged to move the no-GPS zone. The circle to the lower right of the zone can be clicked and dragged to resize and rotate the no-GPS zone.
  - It is possible to define multiple no-GPS zones.

  When you play the scenario, the geolocation does not change from its previous value while the simulated user's position is inside a no-GPS zone. This behavior can be seen in the geolocation widget, which does not update while the simulated user moves through a no-GPS zone.

Figure 3 shows the **Access Point** that requires data to be entered in the fields.



Scenario

Longitude, Latitude (for example -118.15, 34.03) [Go]

Access Point [Delete]

SSID: Network

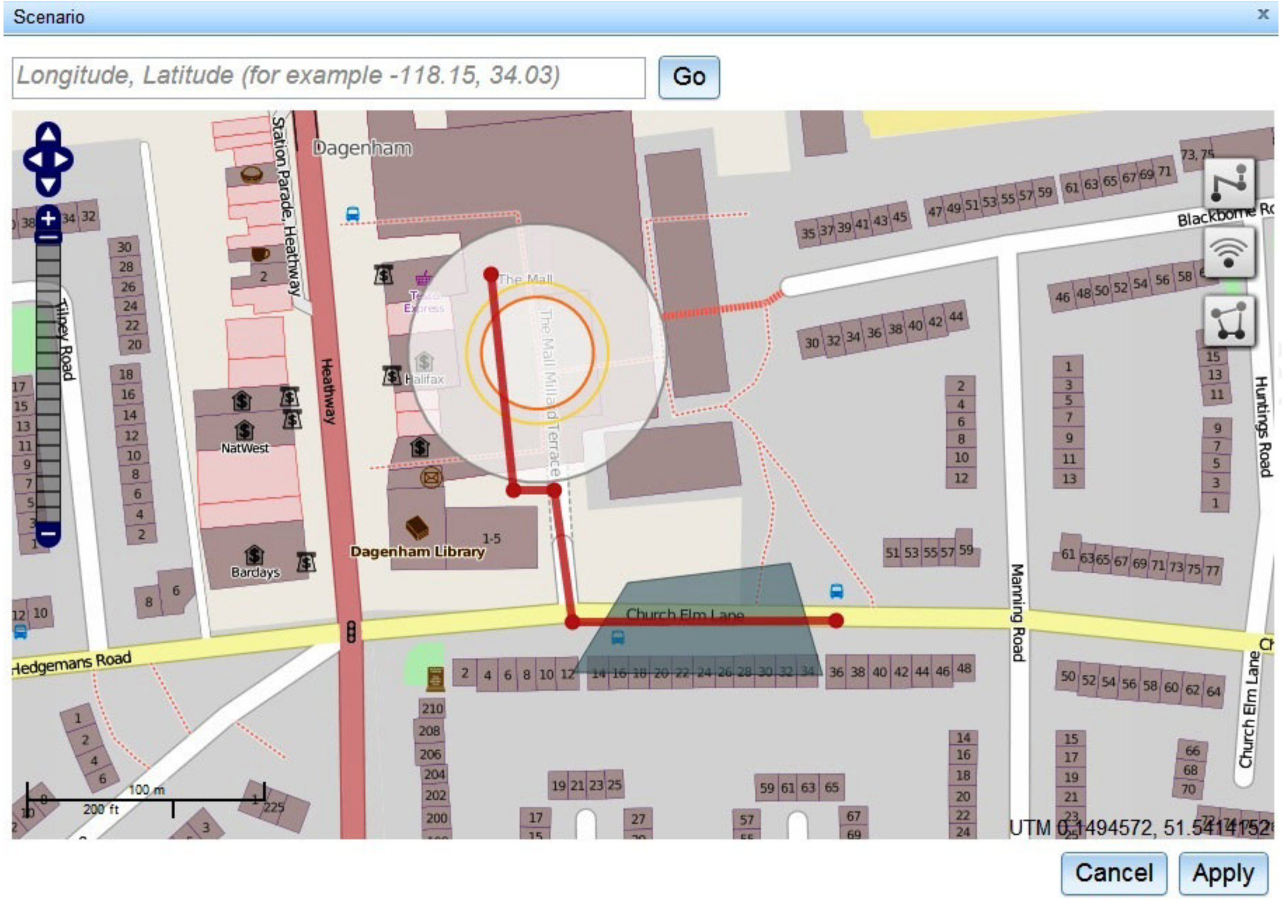
MAC: 14

UTM 0.1515922, 51.5412684

[Cancel] [Apply]

Figure 102.

When you click the WiFi zone, the **Access Point** presents the **SSID** and **MAC** fields, so that you can enter the relevant data. Here you can use the name of a restaurant or store, for example. A **Delete** button is provided if you want to delete the WiFi zone. Two blue circles, or handles, are also visible. These can be used to move or resize the WiFi zone. In figure 4, the WiFi Zone is moved and enlarged. A **no-GPS** zone has been defined, see the gray polygon.



Scenario

Longitude, Latitude (for example -118.15, 34.03)

Go

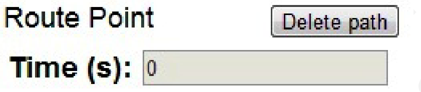
UTM 0.1494572, 51.5414152

Cancel

Apply

Figure 103.

If you click the path, it produces a **Route Point** dialog, which is used to enter how much time it takes the simulated user to reach various points along the route. The first **Route Point** is set at 0, which cannot be edited.

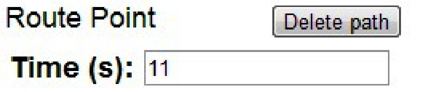


Route Point

Delete path

Time (s): 0

The second **Route Point** can be set as 11, for example

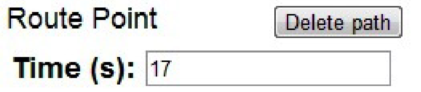


Route Point

Delete path

Time (s): 11

The third **Route Point** can be set as 17, for example



Route Point

Delete path

Time (s): 17

You can change the times, but each value must be increasing from the last. When you play the simulation, the simulated user is at point 1 at time 0, at point 2 at time 11, at point 3 at time 17.

When the scenario is completed, click **Apply** and then **Play** to play the scenario; see Figure 5.

Figure 104. View of Scenario showing the path into the shopping mall.

On your screen you see the user moving along the path, and the information that you receive can be used to interact with the application that is being tested. As the user enters various areas, if a geofence or WiFi trigger is set up then you see that the user's device would register the geofences or WiFi fences.

Click **Export** to save the scenario to disk. You can reload it by clicking **Import**, selecting the file where it was previously saved, and clicking **Play**. After clicking **Play**, the button changes to **Pause** and can be clicked to pause the scenario. Clicking **Restart** plays the scenario from the beginning.

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 the Logger class.

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 699.

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 class.

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 552.

The following example shows an implementation of the `log` procedure in the `WLClientLogReceiver-impl.js` file:

```
function log(deviceInfo, logMessages) {

  /* The adapter can choose to process the parameters,
     for example to forward them to a backend server for
     safekeeping and further analysis.

     The deviceInfo object may look like this:
     {
       "appName":       "wlapp",
       "appVersion":    "1.0",
       "deviceId":      "66eed0c9-ecf7-355f-914a-3cedac70ebcc",
       "model":         "Galaxy Nexus - 4.2.2 - API 17 - 720x1280",
       "systemName":    "Android",
       "systemVersion": "4.2.2",
       "os.arch":       "i686",           // Android only
       "os.version":    "3.4.0-qemu+"     // Android only
      }
      The logMessages parameter is a JSON array
```

```
       that contains JSON object elements, and might look like this:

         [{
           "timestamp"    : "17-02-2013 13:54:23:745",  // "dd-MM-yyyy hh:mm:ss:S"
           "level"        : "ERROR",                    // ERROR || WARN || INFO || LOG || DEBUG
           "package"      : "your_tag",                 // typically a class name, app name, or JavaScript object name
           "msg"          : "the message",              // a helpful log message
           "threadid"     : 42,                         // (Android only) id of the current thread
           "metadata"     : { "$src" : "js" }           // additional metadata placed on the log call
         }]

   */

   return true;

}
```

The procedure element in the WLClientLogReceiver.xml file for log:

```
<procedure name="log" securityTest="wl_unprotected" audit="true" />
```

The security test must be wl_unprotected because apps in the field might be uploading data before the application successfully authenticated. This scenario might occur in the case of crashes during the first time that the app starts.

The audit="true" flag means that uploaded parameters are recorded in the server log. It is a convenient way to get uploaded client logs without having to implement anything in the adapter. You can call WL.Server.log manually in the adapter log procedure implementation.

Client-side logging in client apps

You can take advantage of the client-side logging feature in your client apps.

Log capture

Log capture is enabled by default, but can be turned on or off with native or JavaScript API calls.

Logger native options can be controlled statically by the initOptions.js file in a IBM Worklight generated app. Customers can inspect these values to ensure that they are set as wanted. The processing of initOptions changes the Logger native options. You can affect the log capture configuration programmatically by using the WL.Logger API or statically by specifying the options in the initOptions.js file, but not both at the same time.

Client logs are always captured, but not sent, for first-time start of every IBM Worklight application. To change this behavior, you can specifically place an API call to disable the capture in Android or IOS native code.

Android
The API call to affect the capture setting is:

```
Logger.setCapture(true);
```

iOS The API call to affect the capture setting is:

```
[OCLogger setCapture: YES]
```

JavaScript
The API call to affect the capture setting is:

```
WL.Logger.setNativeOptions({'capture': true});
```

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 690. 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 the Logger class.

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 699.

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 the WL.Logger class.

Polling an adapter or other endpoint to affect logger configuration

You can write client-side code to poll a customer-written adapter. The adapter can reply with an appropriate response, the client must parse the response, and call the appropriate `WL.Logger`, `OCLogger`, or `Logger` API to affect the wanted configuration change.

For example, an adapter implementation might have the following procedure:

```
config() {
  return {capture: true};
}
```

The client code invokes the procedure by using the standard IBM Worklight `invokeProcedure` function as follows:

```
WL.Client.invokeProcedure({
  adapter: 'WLClientLogReceiver',
  procedure: 'config',
  parameters: []
}, {
  onSuccess: function() {
    WL.Logger.setNativeOptions({capture: res.invocationResult.capture}).then(WL.Logger.send);
  }
});
```

You can conditionally return options from the adapter config procedure that is based on some client metadata. To do so, send the metadata through the parameters of the `invokeProcedure` call as follows:

```
environment : WL.Client.getAppProperty(WL.AppProp.ENVIRONMENT)
appName : WL.Client.getAppProperty(WL.AppProp.APP_DISPLAY_NAME)
appVersion : WL.Client.getAppProperty(WL.AppProp.APP_VERSION)
```

Process the incoming data in the adapter config procedure as follows:

```
config(metadata) {
  // turn capture on for Android clients only
  if (metadata.environmen == "android") {
    return {capture: true};
  }
  return {capture: false};
}
```

Chapter 9. API reference

To develop your native or hybrid applications, refer to the IBM Worklight API in JavaScript, Java Platform, Micro Edition (Java ME), Java for Android, and Objective-C for iOS.

Use IBM Worklight API to develop your applications in JavaScript, Java Platform, Micro Edition (Java ME), 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.

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, 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 PCKS#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.

Runtime skinning

Runtime skinning is a feature that helps you incorporate an adaptive design that you can adapt to each mobile device. The IBM Worklight runtime 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.

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.

JavaScript client-side API

You can use JavaScript API to develop apps for all environments.

For more information about these APIs, expand the entry for this topic in the **Contents** panel, and see the *Overview* topic and the *Classes* topic listed there.

The other topics in this section contain additional information that you need to fully use the JavaScript client-side API.

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 107. Options object properties

Property	Description
onSuccess	Optional. The function to be invoked on successful completion of the asynchronous call. The syntax of the onSuccess function is: success-handler-*function*(*response*) where *response* is an object that contains at a minimum the following property: **invocationContext** The invocationContext object that was originally passed to the Worklight Server in the options object, or undefined if no invocationContext object was passed. **status** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.

Table 107. Options object properties (continued)

Property	Description
onFailure	Optional. The function to be invoked when the asynchronous call fails. Such failures include both server-side errors, and client-side errors that occurred during asynchronous calls, such as server connection failure or timed out calls. **Note:** The function is not called for client-side errors that stop the execution by throwing an exception. The syntax of the onFailure function is: failure-handler-*function*(*response*) where *response* is an object that contains the following properties: **invocationContext** The invocationContext object that was originally passed to the Worklight Server in the options object, or undefined if no invocationContext object was passed. **errorCode** An error code string. All error codes that can be returned are defined as constants in the WL.ErrorCode object in the worklight.js file. **errorMsg** An error message that is provided by the Worklight Server. This message is for the developer's use only, and should not be displayed to the user. It will not be translated to the user's language. **status** The HTTP response status **Note:** For methods for which the *response* object contains additional properties, these properties are detailed as part of the description of the specific method.

Table 107. Options object properties (continued)

Property	Description
invocationContext	Optional. An object that is returned to the success and failure handlers. The invocationContext object is used to preserve the context of the calling asynchronous service upon returning from the service. For example, the invokeProcedure method might be called successively, using the same success handler. The success handler needs to be able to identify which call to invokeProcedure is being handled. One solution is to implement the invocationContext object as an integer, and increment its value by one for each call of invokeProcedure. When it invokes the success handler, Worklight passes it the invocationContext object of the options object associated with the invokeProcedure method. The value of the invocationContext object can be used to identify the call to invokeProcedure with which the results that are being handled are associated.

The WL.ClientMessages object

You can see a list of the system messages that are stored in the WL.ClientMessages object, and enable the translation of these system messages.

The WL.ClientMessages object is an object that stores the system messages that are defined in the worklight/messages/messages.json file. This file is in the environment folder of the projects that you generated with IBM Worklight. To enable the translation of a system message, you must override the value of this message in the WL.ClientMessages object, as indicated in the following code example:

```
WL.ClientMessages.invalidUsernamePassword="The custom user name and password are not valid";
```

Note: You must override the system messages on a global JavaScript level because some parts of the code run only after the application successfully initialized.

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.

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.

JavaScript server-side API

The IBM Worklight server-side JavaScript API comprises a series of packages.

For more information about these packages and their content, expand the entry for this topic in the **Contents** panel, and see the *Overview* topic and the *Classes* topic listed there.

Java server-side API

The IBM Worklight server-side Java API comprises a series of packages.

For more information about these packages and their content, 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
ADAPTER_SYNC_DATA	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.
AUTH_ASSOCIATED_IDENTITY	This table is no longer used and might be removed in the future.	n/a
CLUSTER_SYNC	Internal cluster synchronization tasks.	10s of rows.
DEVICES	Tracks devices that access the platform. The devices are recorded as active or inactive, based on the configured decommissioning policy, and other information may be stored for them.	1 row per device that accesses the platform in the last *n* days, where *n* is the sum of the values for the **wl.device.decommission.when** and **wl.device.archiveDecommissioned** parameters.
GADGET_APPLICATIONS	Environments (for example, iPhone, Android) of deployed applications. References the GADGETS table.	10s of rows.
GADGET_USER	This table is no longer used and might be removed in the future.	n/a
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.
LICENSE_TERMS	Stores the various license metrics captured every time the device decommissioning task is run.	10s of rows. Will not exceed the value set by the property **wl.device.decommission.when**.
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.

Name	Description	Order of Magnitude
OPENJPA_SEQUENCE_TABLE	Internal table created for JPA. Not used, and will be removed in the future.	n/a
PROPERTIES	This table is no longer used and might be removed in the future.	n/a
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.
USAGE_DATA	This table is no longer used and might be removed in the future.	n/a
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.
APP_ACTIVITY_REPORT	The reports row data. Data is aggregated by either our aggregation task or by the customer aggregation task. For more information about using the row data, see "Using raw data reports" on page 968.	The size depends on application. The customer is responsible for purging older entries after aggregating to Data Warehouse.

Name	Description	Order of Magnitude
FACT_ACTIVITIES	Summarization of activities that are used for device analytics. 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 wl.db.factProcessingInterval property. The processing and update can also be disabled by setting the wl.db.factProcessingInterval property to a negative value if only the raw data from the APP_ACTIVITY_REPORT table is of interest. For more information about the property, see "Device usage reports" on page 972.	Size depends on app/device usage.
NOTIFICATION_ACTIVITIES	Summarization of activities that are used for notification analytics. Updated with data from the NOTIFICATION_REPORT table. Primarily used by BIRT reports and by other reporting tools.	Size depends on app/notification usage.
NOTIFICATION_PROC_REPORT	Internal table to store raw notification data. The data is aggregated by an aggregation task.	1 row per notification.
NOTIFICATION_REPORT	Each time the data processing is done, a time stamp is added to the PROC_REPORT table with the processing result (timestamp and number of processed entries).	About 72 rows per day.
OPENJPA_SEQUENCE_TABLE	Internal table created for JPA. Not used, and will be removed in the future.	n/a
PROC_REPORT	Internal table that is used for housekeeping and maintaining the state of the scheduler tasks.	About 72 rows per day.

HTTP Interface of the production server

You can use the HTTP interface of the production server to make application API requests or web application resource requests. Use the following request structures, headers, and elements.

Application API requests

Use the following request structure to perform an application API request:

```
{Protocol}://{Worklight Server}/apps/services/api/{Application ID}/{Application Environment}/{Acti
```

Table 108. . Application API request headers

Header Name	Data Type	Description	Valid values
x-wl-app-version	String	Version of the application	
WL-Instance-ID	String	Protection mechanism for XSS attacks.	

Table 109. . Application API request elements

Header Name	Data Type	Description	Valid values
Protocol	String		HTTP
Worklight Server	String	Host name or IP address (and possibly port) identifying the IBM Worklight Server	
Application ID	String	Unique Identifier of the application within the IBM Worklight Server. Every application deployed on the IBM Worklight Server must have a unique identifier	Up to 256 alphanumeric and underscore characters
Application Environment	String	Name of the environment the **application** is running on	air, android, Androidnative, blackberry, desktopbrowser, iOSnative, ipad, iphone, JavaMEnative, mobilewebapp, windows8, windowsphone
Action	String	Requested action	Details in following table

Table 110. . Actions

Action	HTTP Request	Parameters
init	POST	**x, isAjaxRequest** – see the following table showing common parameters.
heartbeat	POST	**x, isAjaxRequest** – see the following table showing common parameters.
logactivity	POST	**x, isAjaxRequest** – see the following table showing common parameters. **activity** – string.
query	POST	**x, isAjaxRequest** – see the following table showing common parameters.**filterList** – **JSON** block **parameterList** – **JSON** block **sorterList** – **JSON** block **Note:** When the action is **query**, the request URL has the following structure: `.../query/{Adapter Name}/{Procedure Name}` where **Adapter Name** and **Procedure Name** are strings.
logout	POST	**x, isAjaxRequest** - see the following table showing common parameters.
login	POST	**x, isAjaxRequest** – see the following table showing common parameters. **realm** – string.
updates	POST	**x, isAjaxRequest** – see the following table showing common parameters. **skin** – current skin name (string) **checksum** – the checksum of the current skin (string) **skinLoaderChecksum** – the checksum of the skin selection code (string)
getup	POST	**x, isAjaxRequest** - see the following table showing common parameters.

Table 110. (continued). Actions

Action	HTTP Request	Parameters
deleteup	POST	**x, isAjaxRequest** – see the following table showing common parameters. **userprefkey** – the user preference to delete.
getuserinfo	POST	**x, isAjaxRequest** – see the following table showing common parameters.
getgadgetprefs	POST	**x, isAjaxRequest** - see the following table showing common parameters.
notifications	POST	**x, isAjaxRequest** – see the following table showing common parameters. **subscribe** – **JSON** string containing subscribe options **unsubscribe** – when specified, designates an unsubscribe action **updateToken** – the update notification token (string) **adapter** – the name of the notification adapter (string) **eventSource** – the name of the notification event source (string) **alias** – notification subscription alias (string)
fbcallback	GET or POST	**x, isAjaxRequest** – see the following table showing common parameters. **popup** – string
composite	POST	**x, isAjaxRequest** - see the following table showing common parameters. **requests** – a **JSON** string containing information about other actions to invoke. This action is used to combine several actions in a single **HTTP** request.
appversionaccess	GET	**x, isAjaxRequest** – see the following table showing common parameters.

Table 110. (continued). Actions

Action	HTTP Request	Parameters
setup	POST	**x, isAjaxRequest** - see the following table showing common parameters. **userprefs** contains JSON pairs of preference key and value
authentication	POST	**x, isAjaxRequest** - see the following table showing common parameters. **action** values are popup, test, or test_img
authenticate	POST	**x, isAjaxRequest** - see the following table showing common parameters. This is an empty handler used to allow the client to respond to authentication challenges with a challengeResponse that cannot fit in a single header or when all headers combined are bigger than the limit for header size.

Table 111. . Common parameters

Parameter	Values	Comments
isAjaxRequest	true	Included with every **GET** and **POST** request only from Adobe™ AIR application.
_	None	Included with every **POST** request only from Webkit-based browsers and application frameworks: Safari, Chrome, and Adobe AIR.

Web application resource requests

Use the following request structure to perform a web application resource request:

```
{Protocol}://{Worklight Server}/apps/services/www/{Application ID}/{Application Environment}/{Applica
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **Application ID**, and **Application Environment**.

Element	Data Type	Description	Valid Values
Application Resource Path	String	HTML, image, JavaScript, CSS, and any other application resource	Example values: img/bg.png, myWidget.html, js/myWidget.js

Preview application resource requests

Use the following request structure to preview application resource requests:

```
{Protocol}://{Worklight Server}/apps/services/preview/{Application ID}/{Application Environment}/{
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **ApplicationID**, and **Application Environment**.

Element	Data Type	Description	Valid Values
Application Resource Path	String	HTML, image, JavaScript, CSS, and any other application resource	Example values: img/bg.png, myWidget.html, js/myWidget.js

Console API requests

Use the following request structure to perform a console API request:

```
http://{hostname}:{port}/{context-root}/console/api/{api-context}/{action}/{parameters}
```

Table 112. Actions

API Context	Action	HTTP Request	Parameters
Adapters	**delete**	POST	**adapterName**
	get	GET	**adapterName**
	all	GET	None
	upload	POST	**adapterName**, **input**

Table 112. Actions (continued)

API Context	Action	HTTP Request	Parameters
Applications	**getPublishUrl**	GET	**gadgetAppId**
	parseCSV	POST	**CSV file**
	delete	POST	**applicationName**
	deleteGadgetApplication	POST	**gadgetAppId**
	setAccessRule	POST	**gadgetAppId** (mandatory), **action** (mandatory: `delete`, `block`, or `notify`), **message** (mandatory), **downloadLink** (optional)
	setAuthenticityRule	POST	**gadgetAppId** (mandatory), **action** (mandatory: `disabled`, `ignored`, or `enabled`)
	setVersionLock	POST	**gadgetAppId**, **lock** (`true` or `false`)
	getBinaryApp	GET	**gadgetAppId**
	all	GET	None
	get	GET	**applicationName**
	upload	POST	**input**, **applicationFolderPath**
Push	**unsubscribeSMS**	POST	**phoneNumbers**
Push, Applications	**all**	GET	None
Push, Mediators	**all**	GET	None
	get	GET	`gcm`, `apns`, or `mpns`
Push, Event Sources	**all**	GET	None
	get	GET	**adapterName/eventSourceName**
Users	**userName**	GET	None
	logout	GET	None

- To retrieve a specific adapter:
  `http://myWorklightServerHost:10080/myProjectRoot/console/api/adapters/get/myAdapterName`
- To retrieve all applications:
  `http://myWorklightServerHost:10080/myProjectRoot/console/api/applications/all`

Chapter 10. 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 `.wlapp` 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.

Important: When building an Android application for deployment to a production environment, do not build it to run in debuggable mode. Ensure that the AndroidManifest.xml file does not include an android:debuggable attribute, or set its value to false. For more information, see Configuring Your Application for Release.

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 331
2. You might want to look at the settings in the worklight.properties file, which is located in server/conf. 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 784.
3. Build each application in either of two ways:
   - Right-click the application and click **Run As** > **Build Settings and Deploy Target**.
   - Use the Ant script tool that is described in "Ant tasks for building and deploying applications and adapters" on page 791

   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. Configure a database and deploy the project WAR to the application server with one of these two methods:
   - With the Worklight Server Configuration Tool. For more information, see "Deploying, updating, and undeploying a Worklight Server by using the Server Configuration Tool" on page 715

- With Ant tasks for configuring a database for a Worklight project and deploying a Worklight project WAR file to an application server. With this method, you can also configure the project on the server using JNDI environment entries.
  - The documentation of the Ant tasks for configuring a database is at “Creating and configuring the databases with Ant tasks” on page 722.
  - The documentation of the Ant tasks for deploying a project WAR file is at “Deploying a project WAR file and configuring the application server with Ant tasks” on page 748.
  - The list of JNDI environment entries that can be configured is at “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784.
  - Sample Ant files using these Ant tasks can be found in the IBM Worklight distribution in `<WORKLIGHT_INSTALL_DIR>/WorklightServer/configuration-samples`. Their file names use the naming convention `configure-<appServer>-<database>.xml`. For more information, see “Sample configuration files” on page 762.

    You must call **configuredatabase** first (target `databases` in the sample Ant files) and then **configureapplicationserver** (target `install` in the sample Ant files).

5. Open the IBM Worklight Console of the target environment. The address is of the format `http://your-remote-server:server-port/context_root/console`

   Note: The context root used in Worklight Server is not necessarily related to the Worklight project name, or to the name of the WAR file created in Worklight Studio. It indicates the common prefix of the path of URLs to the application, as installed on this particular application server.
6. 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 799.
   - 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 794.
7. Obtain the adapters from the development environment.
   a. Navigate to the `bin` folder in your project.
   b. Copy the `.adapter` file or files.
8. 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 799.
   - 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 795.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

Building a project WAR file with Ant

Build the project WAR file with Ant tasks as detailed here.

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 9.

For convenience, Apache Ant 1.8.4 is included in Worklight Server. In the *WL_INSTALL_DIR*/shortcuts/ directory, the following scripts are provided:

- ant for UNIX / Linux
- ant.bat for Windows

These scripts are ready to run, which means that they do not require specific environment variables. If the environment variable JAVA_HOME is set, the scripts accept it.

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-builder.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 the project WAR file

Deploy an IBM Worklight project by following the steps as detailed here.

You need to deploy an IBM Worklight project to an application server to have the Worklight Console available and to have the Worklight Server running. The database and application server prerequisites for this task are described in “Installation prerequisites” on page 52.

You also need a Worklight project WAR file. You can build it by using Worklight Studio, or by following the instructions in “Building a project WAR file with Ant.”

The WAR file contains the Worklight Console, default configuration values for the server, and some resources for the Worklight applications and adapters.

The Project WAR file must be built with the same version of Worklight Studio as the version used to build the apps deployed on the Worklight Server.

You can deploy a Worklight project in the following ways:

- By using the Server Configuration Tool
- By using a set of Ant tasks supplied with Worklight Server to deploy a project WAR file and configure your databases and application servers.
- Follow the procedure for “Creating and configuring the databases manually” on page 732 and for deploying the Worklight Console manually.

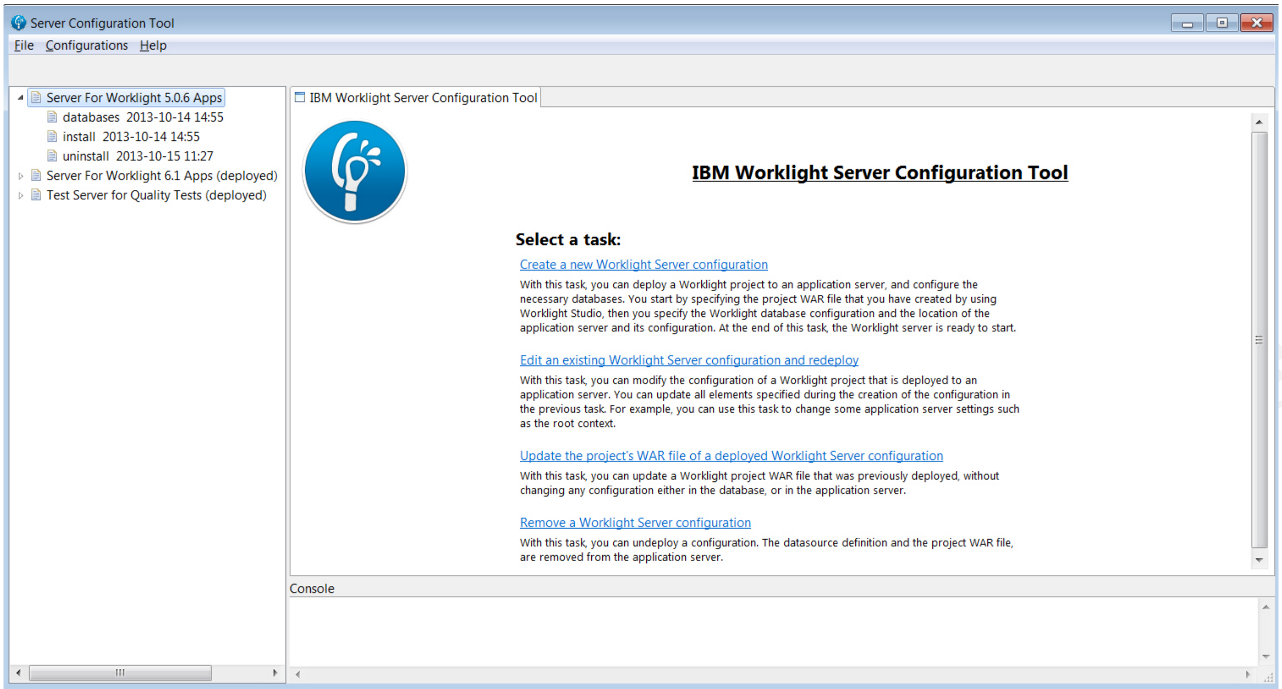
Deploying, updating, and undeploying a Worklight Server by using the Server Configuration Tool

The Server Configuration Tool is a graphical tool that lets you deploy, update, or undeploy a Worklight Server to or from an Application Server and database.

If this tool is used in production to upgrade a Worklight Server, a number of other actions need to be completed to upgrade the server, as described in “Upgrading to Worklight Server V6.1.0 in a production environment” on page 226.

The Server Configuration Tool provides the same capabilities as the Ant tasks described in “Ant tasks for deploying a project WAR file and configuring an application server” on page 718 with the following limitations:

- The tool is limited to four operations (create, edit, update, and remove) described below, that correspond to predefined invocations of the Ant tasks.
- The Derby database is not supported. The supported databases are IBM DB2, Oracle, and MySQL.
- It is not possible to define JNDI deployment properties (such as `publicWorkLightHostname` or other properties listed in “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784). To define those properties, Ant files must be used. You can use the Server Configuration Tool to export an Ant file from a server configuration and then add JNDI deployment properties to it manually. (See “Other operations available in the Server Configuration Tool” on page 718.)
- The Server Configuration Tool must be launched on the computer where your application server is installed.
- The Server Configuration Tool maintains a deployment status of configuration servers (deployed or not). This status is not accurate if the Worklight Server is modified outside the Server Configuration Tool.
- The Server Configuration Tool is available only on Windows and Linux (x86). It is also available on Mac OS for test or demonstration purposes, but the Worklight Server is not supported for production on this platform.



Server Configuration Tool
File Configurations Help
Server For Worklight 5.0.6 Apps
databases 2013-10-14 14:55
install 2013-10-14 14:55
uninstall 2013-10-15 11:27
Server For Worklight 6.1 Apps (deployed)
Test Server for Quality Tests (deployed)
IBM Worklight Server Configuration Tool
Console

IBM Worklight Server Configuration Tool

Select a task:

Create a new Worklight Server configuration

With this task, you can deploy a Worklight project to an application server, and configure the necessary databases. You start by specifying the project WAR file that you have created by using Worklight Studio, then you specify the Worklight database configuration and the location of the application server and its configuration. At the end of this task, the Worklight server is ready to start.

Edit an existing Worklight Server configuration and redeploy

With this task, you can modify the configuration of a Worklight project that is deployed to an application server. You can update all elements specified during the creation of the configuration in the previous task. For example, you can use this task to change some application server settings such as the root context.

Update the project's WAR file of a deployed Worklight Server configuration

With this task, you can update a Worklight project WAR file that was previously deployed, without changing any configuration either in the database, or in the application server.

Remove a Worklight Server configuration

With this task, you can undeploy a configuration. The datasource definition and the project WAR file, are removed from the application server.

Figure 105. Server Configuration Tool main window

Running the Server Configuration Tool

You can start the Worklight Server Configuration Tool in the following ways:

- On Linux:
  - By using the desktop menu shortcut "Server Configuration Tool".
  - In a file manager, click the file *WL_INSTALL_DIR*/shortcuts/configuration-tool.sh.
  - From a shell command line, execute the command *WL_INSTALL_DIR*/shortcuts/configuration-tool.sh.
- On Windows:
  - By using the Start menu shortcut "Server Configuration Tool".
  - In Windows explorer, double-click the file *WL_INSTALL_DIR*/shortcuts/configuration-tool.bat.
  - In a console window, execute *WL_INSTALL_DIR*/shortcuts/configuration-tool.bat.
- On Mac OS X

  Note: The Worklight Server is not supported for production on this platform.
  - In the Finder, double-click the file *WL_INSTALL_DIR*/shortcuts/configuration-tool.sh.
  - In a Terminal window, execute *WL_INSTALL_DIR*/shortcuts/configuration-tool.sh.

Main tasks

Create a new Worklight Server configuration

To create a new Worklight Server configuration, complete the following steps:

1. Enter a descriptive name for the server configuration.

2. Select the path for the Worklight project WAR file to be deployed.
3. Select a shortcut file location. These shortcuts are used to contain the URL of the Worklight Console.
4. Step through the Wizard to describe the target database management system.
   - If you need to create a database for your Worklight Server, the Server Configuration Tool can create it for you if you provide the requested administrator password when prompted in the panel "Database Creation Request". Alternatively, you can ask your database administrator to create the database manually by following the instructions in “Optional creation of databases before you use the Ant tasks” on page 719
5. Follow the steps of the Wizard to describe the target application server. You will need to be authorized to write in the directory of the Application Server.
6. As soon as you have provided all the necessary information, the **Deploy** button is enabled. When you click it, the following actions take place:
   - The configuration file is saved.
   - If the database contains no Worklight tables, the Worklight tables are created.
   - If the database contains Worklight tables for an older version of Worklight, the tables are upgraded to the current Worklight version.
   - If the database operations succeed, the Worklight Server is deployed to the application server. If the WAR file needs to be migrated to the current version, it is migrated.

In the Navigation view, in the left panel of the Server Configuration Tool, log files of the operations are listed under the configuration label.

Edit an existing Worklight Server configuration and redeploy

Use this task to edit and modify an existing Worklight Server configuration. If you select this action, you are prompted to select one of the configurations visible in the Navigation view. If passwords are needed to redeploy the configuration, you are asked to enter them. When you have entered the passwords, the configuration is checked for errors. If errors are found, a report is displayed. You can then edit the configuration. If the configuration contains no errors, the **Redeploy** button is enabled. When you click **Redeploy**, the following actions are completed:

- If the server is already deployed, it is undeployed.
- If undeployment succeeds, the databases are checked. If they contain Worklight tables for an older version of Worklight, the tables are upgraded to the current Worklight version.
- If this succeeds, the new configuration is redeployed. If the WAR file needs to be migrated to the current version, it is migrated.

Note: If you assigned properties to the deployed web application in the application server (for example, security settings), you need to set them again due to the web application having been undeployed.

Update the project's WAR file of a deployed Worklight Server configuration

Use this task to update the WAR file of an existing Worklight Server configuration without changing any other settings. If you select this action, you are asked to select one of the deployed configurations visible in the

Navigation view. If passwords are needed to redeploy the configuration, you are asked to enter them. When you have entered the passwords, the configuration is checked for errors. If errors are found, a report is displayed and the update action is canceled. You can then change the path of the WAR file to be deployed. If your application server is a Liberty or WebSphere Application Server, your new WAR file must have the same name as the WAR file that is already deployed. When you click **Update**, the following action is completed:

- The WAR file in the application server is updated. If necessary, the WAR file is migrated to the current version.

Remove a Worklight Server configuration

Use this task to remove a Worklight Server environment from an application server. If you select this action, you are asked to select one of the deployed configurations visible in the Navigation view. If passwords are needed to remove the configuration from the application server, you are asked to enter them. When you click **Undeploy**, the following actions are completed:

- The Worklight Server is removed from the application server. The database is not modified and the data remains available in the database.

Other operations available in the Server Configuration Tool

Export a Configuration

When you click **File** > **Export**, an Ant task is exported. This Ant file contains tasks that take the following actions:

- Create or update the databases
- Deploy the WAR file
- Update the WAR file
- Undeploy the WAR file

A "`help`" target, the default target of the Ant project, describes the different targets available. You might want to export a configuration for the following reasons:

- To add deployment JNDI properties and then run the Ant file in command line mode with Apache Ant.
- To run the Ant file on a computer without a graphical user interface.
- To perform the Worklight Server operations in batch mode (from the command line and without using a graphical user intarface).

If you modify the Worklight Server status outside the Server Configuration Tool, the status for this configuration is no longer accurate.

Import a Configuration

When you click **File** > **Import**, you can import an Ant file that was generated by the Server Configuration Tool.

Change the working directory where the configurations are stored

From the **File** menu, click **Preferences**, and then select an alternative working directory.

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. For more information about the minimum supported version of Ant, see “System requirements for using IBM Worklight” on page 9.

For convenience, Apache Ant 1.8.4 is included in Worklight Server. In the `WL_INSTALL_DIR/shortcuts/` directory, the following scripts are provided:

- `ant` for UNIX / Linux
- `ant.bat` for Windows

These scripts are ready to run, which means that they do not require specific environment variables. If the environment variable `JAVA_HOME` is set, the scripts accept it.

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 following procedure, 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 projects 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 `~/sqllib/bin` and enter `./db2`.
   - Enter database manager and SQL statements similar to the following example to create the two databases:

     ```
     CREATE DATABASE WRKLGHT COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WRKLGHT
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WRKLGHT
     CREATE DATABASE WLREPORT COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WLREPORT
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WLREPORT
     QUIT
     ```

     Where *wluser* is the name of the system user that you previously created. If you defined a different user name, replace *wluser* accordingly.
3. It is also possible to use only one database (with pagesize settings compatible with what is previously listed), 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;
   ```

   Where *worklight* before the @ sign is the user name, *password* after IDENTIFIED BY is the user password, and *Worklight-host* is 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.

   If the Oracle installation is on a UNIX or Linux machine, make sure that the database will be started the next time the Oracle installation is restarted. To this effect, make sure the line in /etc/oratab that corresponds to the database ends with a Y, not with an N.
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 projects 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* and a user named *WORKLIGHTREPORTS*:

```
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;
```

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 IBM Worklight.

To start the Ant task, you need an Ant XML file with one or more invocations of the `<configuredatabase>` task. The machine on which you run the Ant XML file depends on the type of database:

- For an Apache Derby database, it must be run on the machine that contains the application server.
- For an IBM DB2 or Oracle database, it can be run on any machine. A common choice is to run it on the machine that contains the application server; it will

connect to the database server. It requires SSH access to the database server machine if the database does not yet exist or the user does not have the privileges to use the database.
- For a MySQL database, it can be run on any machine from which the user (or administrator) is allowed to connect to the MySQL server. A common choice is to run it on the machine that contains the application server; it will connect to the database server.

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-deployer.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 719 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 737, 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-deployer.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-deployer.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 732, 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-deployer.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-deployer.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>
      <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-deployer.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-deployer.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 744. 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-deployer.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-deployer.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 does the following:

- Configures a database for a Worklight Project by:
  - Checking if the Worklight tables exist and creating them if needed.
  - If the tables exist and are for an older version of Worklight, migrating them to the current version.
  - If they exist and are for the current version of Worklight, do nothing.
- In addition, if the inner element <dba> is present, the task can:
  - Create the database if needed.
  - Create a user, if necessary, and grant that user privileges to access the database.

The <configuredatabase> task has the following attributes:

Table 113. Attributes for the <configuredatabase> Ant task

Attribute	Description	Required	Default
kind	Type of database (**Worklight** or **WorklightReports**)	Yes	none

It supports the following elements:

Table 114. Inner elements for the <configuredatabase> Ant attribute

Element	Description	Count
derby	Parameters for Derby	0..1
db2	Parameters for DB2	0..1
mysql	Parameters for MySQL	0..1
oracle	Parameters for Oracle	0..1
driverclasspath	JDBC driver class path	0..1

For the configuration of Apache Derby databases

The element <derby> has the following attributes:

Table 115. Attributes for the <derby> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
datadir	Directory that contains the databases	Yes	none
schema	Schema name	No	*WORKLIGHT*

For the configuration of DB2 databases

The element <db2> has the following attributes:

Table 116. Attributes for the <db2> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
server	Host name of database server	Yes	none
port	Port on database server	No	50000
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively
instance	DB2 instance name	No	Depends on server
schema	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 117. Attributes for the <dba> element for DB2 databases

Attribute	Description	Required	Default
user	User name for accessing database	Yes	none
password	Password or accessing database	No	Queried interactively

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 732.

For the configuration of MySQL databases

The element <mysql> has the following attributes:

Table 118. Attributes for the <mysql> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
server	Host name of database server	Yes	none
port	Port on database server	No	3306
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

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 119. Attributes for the <dba> element for MySQL databases

Attribute	Description	Required	Default
user	User name for accessing database	Yes	none
password	Password or accessing database	No	Queried interactively

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 120. Attributes for the <client> element for MySQL databases

Attribute	Description	Required	Default
hostname	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.

Alternatively, you can use the <mysql> element with the following attributes:

Table 121. Alternative attributes for the <mysql> element

Attribute	Description	Required	Default
url	Database connection URL	Yes	none
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

Note: If you specify the database with the alternative attributes, this database must exist, the user account must exist, and the database must already be accessible to the user. In this case, the `<configuredatabase>` task does not attempt to create the database nor the user, nor does it attempt to grant access to the user. The `<configuredatabase>` task only ensures that the database has the required tables for the current Worklight Server version. You do not have to specify the inner elements `<dba>` or `<client>`.

For the configuration of Oracle databases

The element `<oracle>` has the following attributes:

Table 122. Attributes for the <oracle> element

Attribute	Description	Required	Default
database	Database name	No	*ORCL*
server	Host name of database server	Yes	none
port	Port on database server	No	1521
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively
sysPassword	Password for the user SYS	No	Queried interactively, if the database does not yet exist
systemPassword	Password for the user SYSTEM	No	Queried interactively, if the database or the user does not yet exist

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 123. Attributes for the <dba> element for Oracle databases

Attribute	Description	Required	Default
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

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 744.

Alternatively, you can use the `<oracle>` element with the following attributes:

Table 124. Alternative attributes for the <oracle> element

Attribute	Description	Required	Default
url	Database connection URL	Yes	none
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

Note: If you specify the database with the alternative attributes, this database must exist, the user account must exist, and the database must already be accessible to the user. In this case, the `<configuredatabase>` task does not attempt to create the database nor the user, nor does it attempt to grant access to the user. The `<configuredatabase>` task only ensures that the database has the required tables for the current Worklight Server version. You do not have to specify the inner element `<dba>`.

Creating and configuring the databases manually:

You can manually create and configure the IBM Worklight databases.

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 719
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 the following list shows.

Setting up your DB2 databases manually:

You can set up the database manually instead of using the Ant tasks.

About this task

Set up your DB2 database by creating the database schema. The following procedure 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, go to `~/sqllib/bin` and enter `./db2`.
3. Enter the following database manager and SQL statements to create a database that is 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** with the user name.
4. Run 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**, or **password**, or both.

   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 that is called **WLREPORT**:

   ```
   CREATE DATABASE WLREPORT COLLATE USING SYSTEM PAGESIZE 32768
   CONNECT TO WLREPORT
   GRANT CONNECT ON DATABASE TO USER worklight
   QUIT
   ```
6. Run 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 UNIXand 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 768. 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 Using IBM JCC Driver**.
   e. Set **Implementation Type** to **Connection pool data source**.
   f. Set **Name** to **DB2 Using IBM JCC Driver**.
   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”
   - “Configuring WebSphere Application Server for Derby manually” on page 738
   - “Configuring Apache Tomcat for Derby manually” on page 740

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.EmbeddedConnectionPoolDataSou
  <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="1(
  <jdbcDriver libraryRef="DerbyLib"
              javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSou
  <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 768. 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 the 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 the 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 720
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 the following list shows.

Setting up your MySQL databases manually:

You can set up the database manually instead of using the Ant tasks.

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 the option `-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 "at" 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 information 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 768. 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.

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.

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 **Non-transactional data source**.
   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 **Non-transactional data source**. **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** check box.
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 721
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 the following list shows.

Setting up your Oracle databases manually:

You can set up the database manually instead of using the Ant tasks.

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)**.

   If the Oracle installation is on a UNIX or Linux machine, make sure that the database will be started the next time the Oracle installation is restarted. To this effect, make sure the line in /etc/oratab that corresponds to the database ends with a Y, not with an N.

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 768. 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 125. 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. Select **Non-transactional data source**.

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.

The `<configureapplicationserver>` Ant task

You can use the `<configureapplicationserver>` Ant task to 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-deployer.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 722 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.

The Ant task must be run on the same computer as the application server. If you use WebSphere Application Server Network Deployment as the application server, the Ant task must be run on the same machine as the deployment manager.

The `<unconfigureapplicationserver>` Ant task

Use the `<unconfigureapplicationserver>` Ant task to remove the configuration that was done by an earlier `<configureapplicationserver>` invocation. This Ant task takes the same parameters as the `<configureapplicationserver>` invocation.

The `<updateapplicationserver>` Ant task

You can use the `<updateapplicationserver>` Ant task to update the WAR file and, optionally, also update the runtime library in the application server, following an earlier `<configureapplicationserver>` invocation. It does not change the other elements of the application server configuration (such as data sources or JNDI properties).

This Ant task is useful in two situations:
- When you receive an updated project WAR file.
- When you upgrade to a new Worklight Server fix pack; that is, an upgrade in which only the fourth digit of the Worklight Server version has changed.

The `<updateapplicationserver>` Ant task takes the same parameters as the `<configureapplicationserver>` invocation, except that the `<database>` elements may be omitted.

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-deployer.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-deployer.jar"/>
    </classpath>
  </taskdef>

  <target name="databases">
    <configuredatabase kind="Worklight">
      <mysql database="WRKLGHT" server="proddb.example.com" user="wl6test" password="wl6pass">
        <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="wl6test" password="wl6pass">
        <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="wl6test" password="wl6pass"/>
        <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="wl6test" password="wl6pass"/>
        <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="wl6test" password="wl6pass"/>
        <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="wl6test" password="wl6pass"/>
        <driverclasspath>
          <pathelement location="/opt/database-drivers/mysql/mysql-connector-java-5.1.25-bin.jar"/>
        </driverclasspath>
      </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-deployer.jar"/>
    </classpath>
  </taskdef>

  <target name="databases">
    <configuredatabase kind="Worklight">
      <db2 database="WRKLGHT" server="proddb.example.com" user="wl6test" 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="wl6test" 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>

  <target name="install">
    <configureapplicationserver shortcutsDir="/tmp/shortcuts">
      <project warfile="/n/download/testWorklight.war" libraryfile="/n/download/worklight-jee-library

      <!-- 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="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>
    </configureapplicationserver>
  </target>

  <target name="uninstall">
    <unconfigureapplicationserver shortcutsDir="/tmp/shortcuts">
      <project warfile="/n/download/testWorklight.war" libraryfile="/n/download/worklight-jee-libr
      <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 <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> Ant tasks.

The <configureapplicationserver> task configures an application server to run a Worklight project's WAR file as a web application. In detail:

- It declares the Worklight web application in the specified context root (/worklight by default).
- It deploys the project's WAR file on the application server.
- It declares data sources and – on WebSphere Application Server Full Profile – JDBC providers for Worklight and Worklight reports.
- It deploys the Worklight Server runtime (worklight-jee-library.jar) and the database drivers in the application server.
- It configures Worklight configuration properties, through JNDI environment entries. These JNDI environment entries override the Worklight project's default values that are contained in the worklight.properties file inside the WAR file.
- On WebSphere Application Server, it configures a needed web container custom property.

The <updateapplicationserver> task updates an already-configured Worklight web application on an application server. In detail:

- It updates the project's WAR file (which must have the same basename as the project WAR file previously deployed).
- It updates the Worklight Server runtime library (worklight-jee-library.jar).

It does not change the application server's configuration (web application configuration, data sources, JNDI environment entries).

The <unconfigureapplicationserver> task undoes the effects of an earlier <configureapplicationserver> invocation. In detail:

- It removes the configuration of the Worklight web application with the specified context root. This also removes the settings that have been added manually to that application.
- It removes the project's WAR file from the application server.
- It removes the data sources and – on WebSphere Application Server Full Profile – the JDBC providers for Worklight and Worklight reports.
- It removes the Worklight Server runtime library (worklight-jee-library.jar) and the database drivers from the application server.
- It removes the associated JNDI environment entries.

The <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> tasks have the following attributes:

Table 126. Attributes for the <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver>Ant tasks

Attribute	Description	Required	Default
contextroot	Common prefix of path of URLs to the application (context root)	No	/worklight
id	Distinguishes different deployments	No	empty
shortcutsDir	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 backward 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>`, `<updateapplicationserver>`, and `<unconfigureapplicationserver>` tasks support the following elements:

Table 127. Inner elements for the <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> Ant tasks

Element	Description	Count
`project`	Project	1
`property`	Properties	0..∞
`applicationserver`	Application server	1
`reports`	Reports	0..1
`database`	Databases	2

The element `<project>` specifies details about the project to deploy into the application server. It has the following attributes:

Table 128. Attributes for the <project> element

Attribute	Description	Required	Default
`warfile`	Project WAR file	Yes	none
`libraryfile`	File name of `worklight-jee-library.jar`	No	In the same directory as `worklight-ant-deployer.jar`
`migrate`	Whether to auto-migrate the WAR file to the current Worklight Server version	No	true
`migratedWarBackupFile`	Where to store a backup of the migrated WAR file	No	

The `warfile` is created through the `<war-builder>` Ant task. See “Building a project WAR file with Ant” on page 714.

The WAR file is automatically migrated to the current Worklight Server version by default. In this case, you can request that a backup of the migrated WAR file is stored on disk, before it is deployed in the application server. You do this by specifying a value for the **`migratedWarBackupFile`** attribute. If you set migrate to false, migration of the WAR file is not done, and the deployment fails if the Worklight version that produced the WAR file is not suitable for the Worklight Server version.

The element <property> specifies a deployment property that is to be defined in the application server. It has the following attributes:

Table 129. Attributes for the <property> element

Attribute	Description	Required	Default
name	Name of the property	Yes	none
value	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 772.

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 130. Inner elements for the <applicationserver> element

Element	Description	Count
websphereapplicationserver or **was**	Parameters for WebSphere Application Server	0..1
tomcat	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 131. Attributes for the <websphereapplicationserver> or <was> element

Attribute	Description	Required	Default
installdir	WebSphere Application Server installation directory.	Yes	none
profile	WebSphere Application Server profile, or Liberty	Yes	none
user	WebSphere Application Server administrator name	Yes, except for Liberty	none
password	WebSphere Application Server administrator password	No	Queried interactively

It supports the following elements for single-server deployment:

Table 132. Inner elements for the <was> element (single-server deployment)

Element	Description	Count
server	A single server	0..1

The element <server>, used in this context, has the following attributes:

Table 133. Inner elements for the <server> element (single-server deployment)

Attribute	Description	Required	Default
name	Server name	Yes	none

It supports the following elements for Network Deployment:

Table 134. Inner elements for the <was> element (network deployment)

Element	Description	Count
cell	The entire cell	0..1
cluster	All servers of a cluster	0..1
node	All servers in a node, excluding clusters	0..1
server	A single server	0..1

The element <cell> has no attributes.

The element <cluster> has the following attributes:

Table 135. Attributes for the <cluster> element (network deployment)

Attribute	Description	Required	Default
name	Cluster name	Yes	none

The element <node> has the following attributes:

Table 136. Attributes for the <node> element (network deployment)

Attribute	Description	Required	Default
name	Node name	Yes	none

The element <server>, used in a Network Deployment context, has the following attributes:

Table 137. Attributes for the <server> element (network deployment)

Attribute	Description	Required	Default
nodeName	Node name	Yes	none
serverName	Server name	Yes	none

The element <tomcat> denotes an Apache Tomcat server. It has the following attributes:

Table 138. Attributes of the <tomcat> element

Attribute	Description	Required	Default
installdir	Tomcat installation directory. For a Tomcat installation that is split between a CATALINA_HOME directory and a CATALINA_BASE directory, here you need to specify the value of the **CATALINA_BASE** environment variable.	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 139. Attributes of the <reports> element

Attribute	Description	Required	Default
todir	Destination directory	Yes	none

It supports the following elements:

Table 140. Inner elements for the <reports> element

Element	Description	Count
fileset	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 141. Attributes of the <database> element

Attribute	Description	Required	Default
kind	Type of database (*Worklight* or *WorklightReports*)	Yes	none

It supports the following elements:

Table 142. Inner elements for the <database> element

Element	Description	Count
derby	Parameters for Derby	0..1
db2	Parameters for DB2	0..1
mysql	Parameters for MySQL	0..1
oracle	Parameters for Oracle	0..1
driverclasspath	JDBC driver class path	0..1

To specify an Apache Derby database

The element <derby> has the following attributes:

Table 143. Attributes of the <derby> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
datadir	Directory that contains the databases	Yes	none
schema	Schema name	No	*WORKLGHT*

It supports the following elements:

Table 144. Inner elements for the <derby> element

Element	Description	Count
property	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-deployer.jar is used within the installation directory of Worklight.

To specify a DB2 database

The element <db2> has the following attributes:

Table 145. Attributes of the <db2> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
server	Host name of database server	Yes	none
port	Port on database server	No	50000
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

schema	Schema name	No	Depends on user

For more information about DB2 user accounts, see DB2 security model overview.

It supports the following elements:

Table 146. Inner elements for the <db2> element

Element	Description	Count
property	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 147. Attributes of the <mysql> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
server	Host name of database server	Yes	none
port	Port on database server	No	3306
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

A URL can also be specified instead of database, server, and port. The alternative list of attributes is as follows:

Table 148. Alternative elements for the <mysql> element

Attribute	Description	Required	Default
url	Database connection URL	Yes	none
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

For more information about MySQL user accounts, see MySQL User Account Management.

It supports the following elements:

Table 149. Inner elements for the <mysql> element

Element	Description	Count
property	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 150. Attributes of the <oracle> element

Attribute	Description	Required	Default
database	Database name	No	*ORCL*
server	Host name of database server	Yes	none
port	Port on database server	No	1521
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

A URL can also be specified instead of `database`, `server`, and `port`. The alternative list of attributes is as follows:

Table 151. Alternative attributes of the <oracle> element

Attribute	Description	Required	Default
url	Database connection URL	Yes	none
user	User name for accessing database	Yes	none
password	Password for accessing database	No	Queried interactively

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 152. Inner elements for the <oracle> element

Element	Description	Count
property	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 153. Attributes for the <property> element in a database-specific element

Attribute	Description	Required	Default
name	Name of the property	Yes	none
type	Java type of the property's values (usually java.lang.String/Integer/Boolean)	No	java.lang.String
value	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 154. 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 (see Note)	configure-liberty-oracle.xml

Table 154. Sample configuration files provided with IBM Worklight (continued)

Sample	Derby	DB2	MySQL	Oracle
WebSphere Application Server Full profile, single-server	`configure-was-derby.xml`	`configure-was-db2.xml`	`configure-was-mysql.xml` (see Note)	`configure-was-oracle.xml`

WebSphere Application Server Network Deployment	n/a	`configure-wasnd-cluster-db2.xml`
		`configure-wasnd-server-db2.xml`
		`configure-wasnd-node-db2.xml`
		`configure-wasnd-cell-db2.xml`
			`configure-wasnd-cluster-mysql.xml` (see Note)
			`configure-wasnd-server-mysql.xml` (see Note)
			`configure-wasnd-node-mysql.xml` (see Note)
			`configure-wasnd-cell-mysql.xml` (see Note)
				`configure-wasnd-cluster-oracle.xml`
				`configure-wasnd-server-oracle.xml`
				`configure-wasnd-node-oracle.xml`
				`configure-wasnd-cell-oracle.xml`

Apache Tomcat	`configure-tomcat-derby.xml`	`configure-tomcat-db2.xml`	`configure-tomcat-mysql.xml`	`configure-tomcat-oracle.xml`

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. Consider using IBM DB2 or another database supported by WebSphere Application Server to benefit from a configuration that is fully supported by IBM Support.

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 772. 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 install an updated Worklight project in the application server, run the command

```
ant -f configure-file.xml minimal-update
```

To reverse the install step, 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 155. 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 156. Web container custom property values

Property	Value
Name	`com.ibm.ws.webcontainer.invokeFlushAfterService`
Value	`false`
Description	See `http://www.ibm.com/support/docview.wss?uid=swg1PM50111`.

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. Depending on the version of Worklight Studio that was used to build the project WAR file and the version of Worklight Server, you might need to migrate the WAR file first.

When the version of Worklight Studio produces a WAR file that is not compatible with the version of Worklight Server, you must migrate the project WAR file to the current Worklight Server version to ensure a successful manual deployment. All fix packs of a Worklight version are compatible in that sense, and so you do not need to migrate associated project WAR files. For example, if you build a project WAR file by using Worklight Studio V6.0.0 and want to deploy it manually to Worklight Server V6.0.0.1, you do not need to migrate the WAR file. WAR file migration is necessary if Worklight Studio and Worklight Server are of different versions. For example, if you build a project WAR file by using Worklight Studio V6.0.0 and want to deploy it to Worklight Server V6.1.0, you must migrate the WAR file before you deploy it manually. Project WAR files need to be migrated because they contain information that is specific to the Worklight Server version. The migration updates the version-specific information in the WAR file, thus making it suitable to run on the new version of Worklight Server.

Only WAR files produced by Worklight Studio from V5.0.6 and later can be migrated: earlier versions are not supported.

These manual configuration 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.

Migrating a project WAR file for use with a new Worklight Server:

Use Ant tasks to migrate a project WAR file so that you can deploy it manually to a new version of Worklight Server.

You migrate a project WAR file by running a `<migrate>` Ant task, which is included in the `worklight-ant-deployer.jar` library. You can migrate .war files that are developed in Worklight Studio V5.0.6 and later. To run the Ant task, you invoke it from an Ant XML file similar to the following example:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="MigrateWarFile" default="migrate" basedir=".">
  <target name="migrate">
    <echo message="Loading Ant Tool" />
    <taskdef resource="com/worklight/ant/defaults.properties">
      <classpath>
        <pathelement location="worklight-ant-deployer.jar" />
      </classpath>
    </taskdef>
    <migrate sourceWarFile="d:/myOldWarFolder/myProject.war" destWarFile="d:/myNewWarFolder/myMigrate
  </target>
</project>
```

The <migrate> task accepts the following input parameters:

sourceWarFile
(mandatory) The path to the source project WAR file. The path must not contain spaces.

destWarFile
(optional) The destination file for the migrated WAR file. Default value: `<source-war-folder>/migrated-to-<new version number>/<source-war-filename>`.

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 732, 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 of 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 of 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\
     ```
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_conte
   ```

   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 732
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 157. Web container custom property values

Property	Value
Name	`com.ibm.ws.webcontainer.invokeFlushAfterService`
Value	`false`
Description	See `http://www.ibm.com/support/docview.wss?uid=swg1PM50111`

   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 **Select** for `worklight_war` and click **Reference shared libraries**.
   h. From the Available list, select `WL_PLATFORM_LIB` and click **>**.
   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 **Select** for `worklight_war` and click **Reference shared libraries**.
   d. From the Available list, select `WL_PLATFORM_LIB` and click **>**.
   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 **Select** for **worklight_war**and click **Start**.
9. Review the server class loader policy: Click **Servers** > **Server Types** > **Application Servers** > **Worklight**
   - If the class loader policy is set to **Multiple**, do nothing.
   - If the class loader policy is set to **Single** and the class loading mode is set to **parent-last**, do nothing.
   - If the class loader policy is set to **Single** and the class loading mode is set to **parent-first**, change the class loader policy to **Multiple**, and set the class loader order of all applications other than Worklight applications to **parent-first**.

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 732.
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.

Completing the deployment of a project WAR file:

To complete the deployment, you may need to restart the application server.

When the project WAR file is deployed on the application server, you must restart the application server in the following circumstances:

- When you used the `<configureApplicationServer>` Ant task or the manual instructions for deploying the project WAR file:
  - If you are using WebSphere Application Server with DB2 as database type for one or both of the databases.
  - If you are using WebSphere Application Server Liberty Profile or Apache Tomcat.
- When you used the `<updateApplicationServer>` Ant task:
  - If you are using WebSphere Application Server (Full Profile or Liberty Profile) and the Worklight runtime library (`worklight-jee-library.jar`) is changed.
  - If you are using Apache Tomcat.

If you are using WebSphere Application Server Network Deployment and you deployed to managed servers through the deployment manager:

- You must restart the servers that were running during the deployment and on which the Worklight project's web application has been 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.

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 784 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 configuration properties” on page 781 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 158. Worklight Server location properties

Property name	Description
publicWorkLightHostname	The IP address or host name of the computer running IBM Worklight. If the Worklight Server is behind a reverse proxy, the value is the IP address or host name of the reverse proxy. This property must be identical for nodes within the same cluster. Default: IP address of current server.
publicWorkLightPort	The port for accessing the Worklight Server. If the 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: `10080`. The <configureApplicationServer> Ant task sets a default value that depends on the application server.
publicWorkLightProtocol	The protocol for accessing the Worklight Server. 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. This property must be identical for nodes within the same cluster. Default: `HTTP`. The <configureApplicationServer> Ant task sets a default value that depends on the application server.

For descriptions of other configuration properties, see “Configuring an IBM Worklight project in production by using JNDI environment entries” on page 784

For information about how to specify configuration properties, see “Configuration of IBM Worklight applications on the server” on page 772.

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 732.

Property keys and values for JDBC-based properties

Property Key	Property Value
`wl.db.url`	JDBC path to IBM Worklight database.
`wl.db.username`	IBM Worklight database user name. Default: `Worklight`
`wl.db.password`	IBM Worklight database password. Default: `Worklight`
`wl.db.driver`	The class that implements a JDBC driver for each vendor. For example: MySQL: `com.mysql.jdbc.Driver` Oracle: `oracle.jdbc.OracleDriver` DB2: `com.ibm.db2.jcc.DB2Driver` Derby: `org.apache.derby.jdbc.EmbeddedDriver`
`wl.reports.db.url(*)`	JDBC path to IBM Worklight Reports database Default: refers to IBM Worklight database
`wl.reports.db.username(*)`	IBM Worklight Reports database user name. Default: refers to IBM Worklight database
`wl.reports.db.password(*)`	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 968.

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 779.

Property Key	Property Value
`console.username`	Name of the user that can access the Console
`console.password`	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 securityTes
  <staticResources>
  <!--
  <resource id="worklightConsole" securityTest="WorklightConsole">
    <urlPatterns>/console*</urlPatterns>
  </resource>
  -->

<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 608.

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
push.gcm.proxy.enabled	Shows whether Google GCM must be accessed through a proxy. Can be either true or false. The default is false.
push.gcm.proxy.protocol	GCM proxy protocol. Can be either http or https.
push.gcm.proxy.host	GCM proxy host.
push.gcm.proxy.port	GCM proxy port. Use -1 for the default port.
push.gcm.proxy.user	Proxy user name, if the proxy requires authentication. An empty user name means no authentication.
push.gcm.proxy.password	Proxy password, if the proxy requires authentication.

APNS proxy settings	Value
push.apns.proxy.enabled	Shows whether APNS must be accessed through a proxy. Can be either true or false. The default is false.
push.apns.proxy.type	APNS proxy type. Must be SOCKS.
push.apns.proxy.host	APNS proxy host.

APNS proxy settings	Value
push.apns.proxy.port	APNS proxy port.

SMS proxy settings	Value
push.sms.proxy.enabled	Can be either true or false. Use true to send SMS notifications through proxy.
push.sms.proxy.protocol	SMS proxy protocol. Can be either http or https.
push.sms.proxy.host	SMS proxy host name.
push.sms.proxy.port	SMS proxy port. Use -1 for the default port.
push.sms.proxy.user	Proxy user name, for authentication. An empty user name means no authentication.
push.sms.proxy.password	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 IBM SmartCloud Analytics Embedded (operational analytics) server by editing the following properties files.

Table 159.

Property Key	Property Value
wl.analytics.logs.forward	A Boolean value (true or false) that indicates whether to send messages that are logged by the Worklight Server to the operational analytics server. If this value is true, all logs that are specified in com.worklight settings are forwarded to the operational analytics server. The default value is true. This setting is only supported on IBM Worklight production servers. It s not supported on the IBM Worklight Studio development environment. (Optional.)
wl.analytics.url	The url to which analytics data are sent. Typically, this url is for the operational analytics server that comes with the IBM Worklight installation. The URL is of the form <protocol>://<iwapURL>/iwap/v1/events/_bulk Example: http://myServer.austin.ibm.com/iwap/v1/events/_bulk **Note:** The **Analytics** tab is displayed in the IBM Worklight console only if the user enters a valid link for wl.analytics.url in the worklight.properties file. If the properties file contains no URL, the **Analytics** tab is not present.
wl.analytics.username	The basic authorized user name for the URL entered in wl.analytics.url. (Optional.)

Table 159. (continued)

Property Key	Property Value
wl.analytics.password	The basic authorized password for the URL entered in `wl.analytics.url`. (Optional.)
wl.analytics.queues	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).
wl.analytics.queue.size	Sets the number of individual analytics events that each queue can hold. The total number of analytics events that the server can hold at one time before it begins to drop data is (`wl.analytics.queues * wl.analytics.queue.size`). In a production environment, the default value is `10`. In the Worklight Studio development environment when you use the Worklight Development Server, the default value is 1. This value can be changed by setting a different value through JNDI. (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 160. JNDI environment entries for SSL certificate keystore

Property name	Description
ssl.keystore.path	Path to the keystore relative to the server folder in the Worklight Project; for example: `conf/my-cert.jks`.
ssl.keystore.type	Type of keystore file. Valid values are `jks` or `pkcs12`.
ssl.keystore.password	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 784

For information about how to specify IBM Worklight configuration properties, see “Configuration of IBM Worklight applications on the server” on page 772.

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 529.

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 534.

  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
`serverSessionTimeout`	Client inactivity timeout, after which the IBM Worklight session is invalidated. Default is 10 minutes.
`bitly.username`	User name for accessing the bit.ly API for creating a shortened URL for mobile web apps through IBM Worklight Console.
`bitly.apikey`	The bit.ly API Key.
`compress.response.threshold`	The threshold size of the payload that is returned in response to an `invokeProcedure` call beyond which the response is compressed. The default value is 20480 bytes. Responses with payload larger than the **`compress.response.threshold`** are compressed by the server. To disable compression, set this value to a large value. Similarly, to compress every response, set this value to 0 (zero). If the payload is larger than the **`compress.response.threshold`**, the payload is compressed irrespective of whether or not compression was requested by the client through the `compressResponse` option.

Storing properties in encrypted format

You must encrypt the properties that are too sensitive to be written in clear text.

There are 2 ways to encrypt the properties:

- “Encryption within the properties file.” This is the only option available with Tomcat.
- For WebSphere Application Server and Liberty Profile: “Encoding the JNDI properties” on page 781 by using the application server encoding tools (`PropFilePasswordEncoder` for WebSphere Application Server and `SecurityUtility` for Liberty Profile).

Encryption 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 open or encrypted format

You can keep properties contained in `worklight.properties` either in open or in encrypted form.

An encrypted property is determined by a suffix `.enc` on its name, for example:

```
console.password.enc=TYakEHRba3rIU7pNjxtDxoAdqijKIEt7cy4mCr0iaEj0rY08ODK00yqR
```

The IBM Worklight configuration is accessed for a property. If the property is not found, but the same encrypted property (with `.enc` suffix) is defined, IBM Worklight automatically decrypts the value and returns it to the caller.

Storing the master key

All of the encrypted values use the same secret key, which is stored in the special variable called **`worklight_enc_password`**. This variable is defined as an operating system environment variable:

- On Windows systems: Set an environment variable under the user running 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=dR4lnMQDaNEQyLQl7b2RmpdE99HKpqaSJ6mce0uJgaY\=
wl.db.password.enc=6boxojGZsUNTXwOOGgI6dg\=\=
```

Encoding the JNDI properties

The preferred way to encrypt the JNDI properties in WebSphere Application Server is to use the password encoding tools available with both products:

- The `PropFilePasswordEncoder` tool for WebSphere Application Server.
- The `SecurityUtility` command for Liberty Profile.

The encoded value can then be used for the value of the JNDI properties.

For more details about how to encode the properties with the application server tools, see the WebSphere Application Server documentaton.

Obsolete properties

Some properties are no longer required.

Table 161. Categories and list of obsolete properties

Category	Properties
Proxy settings	**`proxy.enabled`**, **`proxy.nonProxyHosts`**, **`proxy.host`**, **`proxy.port`**, **`proxy.username`**, **`proxy.password`**, **`https.proxy.host`**, **`https.proxy.port`**
Public resource server settings	**`publicResourceServer.deployDestination`**, **`publicResourceServer.host`**, **`publicResourceServer.port`**, **`publicResourceServer.filesRootDir`**
Environments	**`environment.netvibes`**, **`environment.iphone`**, **`environment.embedded`**, **`environment.air`**, **`environment.android`**, **`environment.blackberry`**
Certificate settings	**`certificate.certificatesDirPath`**, **`certificate.keyStoreFilePath`**, **`certificate.keyAlias`**, **`certificate.keyStorePassword`**, **`certificate.keyAliasPassword`**, **`certificate.PFXFilePath`**, **`certificate.password`**, **`certificate.DERFilePath`**, **`certificate.P7BFilePath`**, **`vista.linux.osslsigncodeFilepath`**
Push notification settings	**`push.apns.certificatePassword`**, **`push.gcm.senderID`**, **`push.gcm.senderPassword`**
Miscellaneous settings	**`devmode`**, **`guid`**, **`wlclientTimeout`**, **`backend.request.timeout`**, **`reports.produceReports`**, **`wl.db.initialSize`**, **`wl.db.maxActive`**, **`wl.db.maxIdle`**, **`wl.db.testOnBorrow`**, **`wl.db.autoddl`**
Tomcat settings	**`local.bindAddress`**, **`local.httpPort`**
Console security settings	**`console.username`**, **`console.password`**

Declaring and using configuration properties

Use the `${propertyName}` notation to reuse project-specific properties that are declared in the `worklight.properties` file.

You can declare project-specific properties in the `worklight.properties` file. You can then reuse the value of those properties within the authentication configuration file (`authenticationConfig.xml`) and the adapter descriptor file (`adapter.xml`) by using the `${propertyName}` notation.

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" on page 784) 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.

In `authenticationConfig.xml`, you can use `${propertyName}` notation for all realm and **`loginModule`** parameters. Here are examples in bold for such properties:

```
<securityTests>

  <customSecurityTest name="MySecurityTest">
    <test realm="MySecurityRealm" isInternalUserID="true"/>
  </customSecurityTest>

</securityTests>

<realms>

  <realm name="MySecurityRealm" loginModule="MySecurityLoginModule">
    <className>com.test.auth.MyAuthenticator</className>
    <parameter name="login-mode" value="${my.security.realm.mode}"/>
    <parameter name="my-other-realm-param" value="${my.security.realm.param}"/>
 </realm>

</realms>

<loginModules>

  <loginModule name="MySecurityLoginModule">
    <className>com.test.auth.MyLoginModule</className>
    <parameter name="roles-allowed" value="${my.security.allowed.roles}"/>
    <parameter name="my-other-login-param" value="${my.security.login.param}"/>
  </loginModule>

 </loginModules>
```

For more information about configuring realm parameters, see "Configuring authenticators and realms" on page 610. For **`loginModule`** parameters, see "Configuring login modules" on page 616.

In `adapter.xml`, you can use the `${propertyName}` notation in the following elements:

For HTTP adapters:

```
<connectivity>
   <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
    <protocol>${my.protocol}</protocol>
     <domain>${my.domain}</domain>
     <port>${my.port}</port>

    <authentication>
      <ntlm workstation="${local.hostname}" />
      <serverIdentity>
        <username>${my.server.identity.username}</username>
        <password>${my.server.identity.password}</password>
      </serverIdentity>
    </authentication>

    <!-- Following properties used by adapter's key manager for choosing specific certific

    <sslCertificateAlias>${my.ssl.certificate.alias}</sslCertificateAlias>
    <sslCertificatePassword>${my.ssl.certificate.password}</sslCertificatePassword>

  </connectionPolicy>

  <loadConstraints maxConcurrentConnectionsPerNode="${max.connections.per.node}"/>
</connectivity>
```

For SQL adapters:

```
<connectivity>
  <connectionPolicy xsi:type="sql:SQLConnectionPolicy">

  <!-- Example for using a JNDI data source, replace with actual data source name -->
  <!-- <dataSourceJNDIName>${my.data.source.jndi.name}</dataSourceJNDIName> -->

  <!-- Example for using MySQL connector, do not forget to put the MySQL connector library
  <dataSourceDefinition>
    <driverClass>${my.driver.class.name}</driverClass>
    <url>${my.data.source.url}</url>
    <user>${my.data.source.username}</user>
    <password>${my.data.source.password}</password>
  </dataSourceDefinition>
</connectionPolicy>
  <loadConstraints maxConcurrentConnectionsPerNode="${max.connections.per.node}" />
</connectivity>
```

For JMS adapters:

```
<connectivity>
  <connectionPolicy xsi:type="jms:JMSConnectionPolicyType">

  <!-- uncomment this if you want to use an external JNDI  repository -->
  <!-- <namingConnection url="${my.naming.connection.url}"
    initialContextFactory="${my.initial.context.factory}"
    user="${my.naming.connection.username}"
    password="${my.naming.connection.password}"/>
   -->

    <jmsConnection connectionFactory="${my.jms.connection.factory}"
      user="${my.jms.connection.username}"
      password="${my.jms.connection.password}"
    />
  </connectionPolicy>

  <loadConstraints maxConcurrentConnectionsPerNode="${max.connections.per.node}"/>
</connectivity>
```

For more information about configuring adapters, see "The <authentication> element of the HTTP adapter" on page 535.

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. These entries 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 are used to specify the Worklight Server public URL (that is, **`publicWorkLightHostname`**, **`publicWorkLightPort`**, and **`publicWorkLightProtocol`**) 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: There are two ways to encrypt the JNDI properties that are listed in the following table, as described in "Storing properties in encrypted format" on page 779:

- You can 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. This is the only option available with Apache Tomcat.
- On WebSphere Application Server and Liberty Profile, you can use the password encoding tools (`PropFilePasswordEncoder` for WebSphere Application Server and `SecurityUtility` for Liberty Profile).

The following table lists the IBM Worklight properties that are always available as JNDI entries:

Table 162. IBM Worklight properties available as JNDI entries

Property name	Description
`publicWorkLightHostname`	The IP address or host name of the computer that is running IBM Worklight. If the 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.

Table 162. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
publicWorkLightPort	The port for accessing the Worklight Server. If the 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: `10080`. The <configureApplicationServer> Ant task sets a default value that depends on the application server.
publicWorkLightProtocol	The protocol for accessing the Worklight Server. 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. This property must be identical for nodes within the same cluster. Default: `HTTP`. The <configureApplicationServer> Ant task sets a default value that depends on the application server.
serverSessionTimeout	Idle session timeout in minutes. Default: `10`.
reports.exportRawData	Whether reporting is activated (`true` or `false`). Default: `false`.
push.gcm.proxy.host	GCM proxy host. A negative value means default port.
push.gcm.proxy.port	GCM proxy port. Use `-1` for the default port. Default: `-1`.
push.gcm.proxy.protocol	Either `http` or `https`.
push.gcm.proxy.enabled	Shows whether GCM must be accessed through a proxy. Default: `false`.
push.gcm.proxy.user	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
push.gcm.proxy.password	Proxy password, if the proxy requires authentication.
push.apns.proxy.enabled	Indicates whether APNS must be accessed through a proxy. Default: `false`.
push.sms.proxy.enabled	Indicates whether push SMS proxy is enabled. Default: `false`.
push.apns.proxy.host	APNS proxy host.
push.apns.proxy.port	APNS proxy port.
push.sms.proxy.protocol	Push SMS proxy protocol.

Table 162. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
push.sms.proxy.host	Push SMS proxy host.
push.sms.proxy.port	Push SMS proxy port.
push.sms.proxy.user	Push SMS proxy user.
push.sms.proxy.password	Push SMS proxy password.
wl.ca.keystore.path	Path to the keystore relative to the server folder in the Worklight Project; for example: `conf/my-cert.jks`.
wl.ca.keystore.type	Type of keystore file. Valid values are `jks` or `pkcs12`.
wl.ca.keystore.password	Password to the keystore file.
wl.ca.key.alias	Alias of the entry where the private key and certificate are stored in the keystore.
wl.ca.key.alias.password	Password to the alias in the keystore.
ssl.keystore.path	SSL certificate keystore location. Default: `conf/default.keystore`.
ssl.keystore.type	SSL certificate keystore type. Valid keystore types: `jks` or `PKCS12`. Default: `jks`.
ssl.keystore.password	SSL certificate keystore password. Default: `worklight`.
cluster.data.synchronization.taskFrequencyInSeconds	Applications and adapters cluster data synchronization interval. Default: `2`.
deployables.cleanup.taskFrequencyInSeconds	Deployable folder cleanup task interval (in seconds). Default: `86400`.
sso.cleanup.taskFrequencyInSeconds	Interval (seconds) for a cleanup task that cleans the database of orphaned and expired single-sign-on login contexts. Default: `5`
wl.analytics.logs.forward	Boolean value (`true` or `false`) that indicates whether to send all `com.worklight.*` logs to the operational analytics server. If this value is `true`, all logs that are specified in com.worklight settings are forwarded to the operational analytics 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.
wl.analytics.url	URL to which analytics data are sent. Typically, this URL is for the operational analytics 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 `_bulk`. Example: `http://myServer.austin.ibm.com/iwap/v1/events/_bulk`.

Table 162. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
`wl.analytics.username`	Basic authorized user name for the URL entered in `wl.analytics.url`. **Note:** The user name is only required if you configured the IBM HTTP Server (IHS) in your IBM SmartCloud Analytics Embedded installation to use SSL.
`wl.analytics.password`	Basic authorized password for the URL entered in `wl.analytics.url`. **Note:** The password is only required if you configured the IBM HTTP Server (IHS) in your IBM SmartCloud Analytics Embedded installation to use SSL.
`wl.analytics.queues`	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`.
`wl.analytics.queue.size`	Number of individual analytics events that each queue can hold. The total number of analytics events that the server can hold at one time before it begins to drop data is (`wl.analytics.queues` * `wl.analytics.queue.size`). In a production environment, the default value is `10`. In the Worklight Studio development environment, when you use the Worklight Development Server, the default value is 1. This value can be changed by setting a different value through JNDI. (Optional.)
`wl.device.archiveDecommissioned.when`	A value, in days, that defines when client devices that were decommissioned will be placed in an archive file when the decommissioning task is run. The archived client devices are written to a file in the Worklight Server `home\devices_archive` directory. The name of the file contains the time stamp when the archive file is created. Default: `90 days`.
`wl.device.decommission.when`	The number of days of inactivity after which a client device is decommissioned by the device decommissioning task. Default: `90 days`.
`wl.device.enableAccessManagement`	A Boolean value (`true` or `false`) that enables the Access Management features on the Worklight Server. If the Access Management features are enabled, each time a device attempts to connect to the server, it is checked against the backend for its access rights.

Table 162. IBM Worklight properties available as JNDI entries (continued)

Property name	Description
wl.device.tracking.enabled	A value that is used to enable or disable device tracking in Worklight. For performance reasons, you can disable this flag when Worklight is running only Business-to-Consumer (B2C) apps. When device tracking is disabled, the license reports are also disabled and no license metrics are generated.

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 748), 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"/>
```

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, you can define specific JNDI entries for each application by using the context path prefix. Typically, **app_context_path** is **"worklight"**.
- Apache Tomcat:

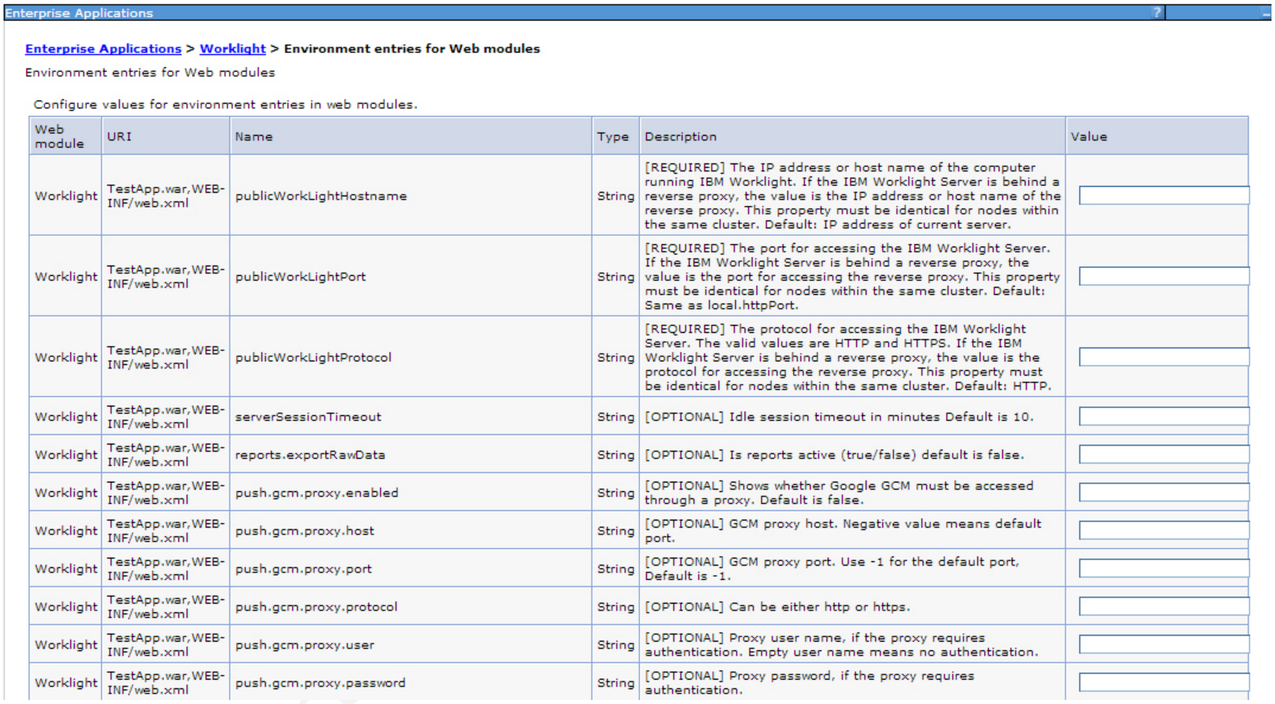
Insert the following declarations in the server.xml file:

```
<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, go to **Applications** > **Application Types** > **WebSphere enterprise applications** > **Worklight** > **Environment entries for Web modules**
  2. In the **Value** fields, enter values that are appropriate to your circumstances. See Figure 106



Enterprise Applications

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.	
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.	
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.	
Worklight	TestApp.war,WEB-INF/web.xml	serverSessionTimeout	String	[OPTIONAL] Idle session timeout in minutes Default is 10.	
Worklight	TestApp.war,WEB-INF/web.xml	reports.exportRawData	String	[OPTIONAL] Is reports active (true/false) default is false.	
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.	
Worklight	TestApp.war,WEB-INF/web.xml	push.gcm.proxy.host	String	[OPTIONAL] GCM proxy host. Negative value means default port.	
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.	
Worklight	TestApp.war,WEB-INF/web.xml	push.gcm.proxy.protocol	String	[OPTIONAL] Can be either http or https.	
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.	
Worklight	TestApp.war,WEB-INF/web.xml	push.gcm.proxy.password	String	[OPTIONAL] Proxy password, if the proxy requires authentication.	

Figure 106. Setting JNDI environment entries on WebSphere Application Server

Related reference:

"Configuration of IBM Worklight applications on the server" on page 772
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 163. `SMSConfig.xml` elements and subelements

Element	Element Value
`gateway`	Mandatory. The `<gateway>` element is the root element of the SMS gateway definition. It includes 6 attributes: • `hostname` • `id` • `port` • `programName` • `toParamName` • `textParamName` These attributes are described in Table 164
`parameter`	Optional. The `<parameter>` subelement is dependent on the SMS gateway. Each SMS gateway may have its own set of parameters. The number of `<parameter>` subelements is dependent on SMS gateway-specific parameters. If an SMS gateway requires the user name and password to be set, then these parameters can be defined as `<parameter>` subelements. Each `<parameter>` subelement has the following attributes: • `name` • `value`

Table 164. `<gateway>` element attributes

Attribute	Attribute Value
`hostname`	Mandatory. The host name of the configured SMS gateway.
`id`	Mandatory. A unique ID that identifies the SMS gateway. Application developers specify the ID in the application descriptor file, `application-descriptor.xml`, when they develop an application.
`port`	Optional. The port number of the SMS gateway. The default value is 80.
`programName`	Optional. The name of the program that the SMS gateway expects. For example, if the SMS gateway expects the following URI: `http://<hostname>:port/sendsms` then **`programName`**="sendsms"

Table 164. <gateway> element attributes (continued)

Attribute	Attribute Value
toParamName	Optional. The name that is used by the SMS gateway to specify the destination mobile phone number. The default value is *to*. The destination mobile phone number is sent as a name-value pair when SMS notifications are sent; that is, *toParamName=destination mobile phone number*.
textParamName	Optional. The name that is used by the SMS gateway to specify the SMS message text. The default value is *text*.

If the SMS gateway expects an HTTP post in the following format to forward SMS messages to a mobile device:

`http://myhost:13011/cgi-bin/sendsms?to=`*`destination mobile phone number`*`&text=`*`message text`*`&username=fcsuser&password=fcspass`

The `SMSConfig.xml` file is configured as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<sms:config xmlns:sms="http://www.worklight.com/sms/config" xmlns:xsi="http://www.w3.org/2001/XMLS

  <gateway hostname="myhost" id="kannelgw" port="13011" programName="cgi-bin/sendsms" toParamName=
   <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 9.

For convenience, Apache Ant 1.8.4 is included in Worklight Server. In the *WL_INSTALL_DIR*/shortcuts/ directory, the following scripts are provided:

- ant for UNIX / Linux
- ant.bat for Windows

These scripts are ready to run, which means that they do not require specific environment variables. If the environment variable JAVA_HOME is set, the scripts accept it.

Building from a Worklight V6.0.0 project and deploying to a V6.1.0 Worklight Server

If you want to build apps and adapters from a Worklight Studio V6.0.0 project and deploy them to a V6.1.0 Worklight Server, you might think that all you need to do is add an additional <taskdef> definition as shown in the following Ant task:

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="WL600_DIR/WorklightServer/worklight-ant.jar" />
  </classpath>
</taskdef>
```

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="WL610_DIR/WorklightServer/worklight-ant-deployer.jar" />
  </classpath>
</taskdef>
```

Note: `WL600_DIR` and `WL610_DIR` are the installation directories of Worklight Server.

However, the definitions of `<app-deployer>` and `<adapter-deployer>` conflict in the aforementioned Ant script. To solve this conflict, you must split the script into two different Ant files: one to build V6.0.0 artifacts, and the other to deploy them to the V6.1.0 server, as shown in the following examples:

Ant script to build V6.0.0 artifacts

```
<project basedir="." default="build-and-deploy">

  <property name="project.name" value="MyProject" />
  <property name="wl.server" value="http://localhost:9080/${project.name}/" />
  <property name="wl.project.location" location="${basedir}/${project.name}" />
  <property name="output.location" location="${wl.project.location}/bin" />

  <property name="wl.adapter.name" location="MyAdapter" />
  <property name="wl.application.name" location="MyApplication" />

  <property name="worklight-ant" location="worklight-ant.jar" />

  <target name="init">
    <taskdef resource="com/worklight/ant/defaults.properties">
      <classpath>
        <pathelement location="${worklight-ant}" />
      </classpath>
    </taskdef>
  </target>

  <target name="build">
    <adapter-builder folder="${wl.project.location}/adapters/${wl.adapter.name}" destinationI
    <app-builder applicationFolder="${wl.project.location}/apps/${wl.application.name}" outpu
  </target>

  <target name="deploy">
    <ant antfile="deploy.xml" inheritall="true" />
  </target>

  <target name="build-and-deploy" depends="init,build,deploy" />
</project>
```

Ant script to deploy V6.0.0. artifacts to a V6.1.0 server

```
<project basedir="." default="deploy">

  <property name="worklight-ant-deployer" location="worklight-ant-deployer.jar" />

  <target name="init">
    <taskdef resource="com/worklight/ant/defaults.properties">
```

```
        <classpath>
          <pathelement location="${worklight-ant-deployer}" />
        </classpath>
      </taskdef>
    </target>

    <target name="deploy" depends="init">
      <adapter-deployer deployable="${output.location}/${wl.adapter.name}.adapter" worklight
        <app-deployer deployable="${output.location}/${wl.application.name}-all.wlapp" workl
    </target>
  </project>
```

Building applications and adapters

The Ant tasks used for building IBM Worklight applications and adapters 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-builder.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <app-builder
        worklightserverhost="http://server-address:port"
        applicationFolder="application-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 `ouptputFolder` attribute specifies the folder to which the resulting `.wlapp` file is written.

By default, running the Ant task to build an application does not handle the Dojo Toolkit, because Ant is not run with `build-dojo.xml`. You must explicitly configure the task to do so, by using the following app-builder setting in the Ant build file:

```
skinBuildExtensions=build-dojo.xml
```

If you use this setting, the Dojo Toolkit files are deployed with your application.

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-builder.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 applications and adapters

The Ant tasks used for deploying IBM Worklight applications and adapters are documented in this section.

The following sections show examples of Ant XML files that use the <app-deployer> Ant task to deploy applications and the <adapter-deployer> Ant task to deploy adapters. These Ant tasks can be run locally on the Worklight Server host computer or remotely on a different computer. To run them remotely on a different computer, you must first copy the file *<WorklightInstallDir>*/WorklightServer/worklight-ant-deployer.jar to that computer.

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 the project WAR file" on page 714.

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-deployer.jar" />
    </classpath>
  </taskdef>
  <target name="target-name">
    <app-deployer deployable="app.wlapp"
                  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 .wlapp 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 .wlapp file, add an <app-deployer> element for each file.

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 the project WAR file" on page 714.

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-deployer.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 .wlapp 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 791 and “Ant tasks for deploying a project WAR file and configuring an application server” on page 718.
- Use Worklight Console to manually deploy apps and adapters as described in the section.

Worklight Console opens in a “Catalog” page that enables you to work with apps and adapters.

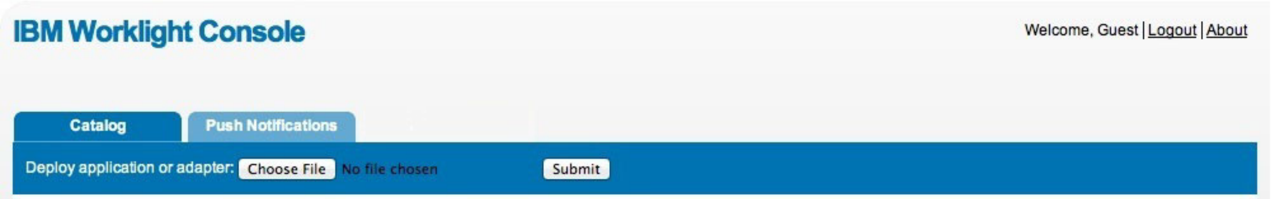


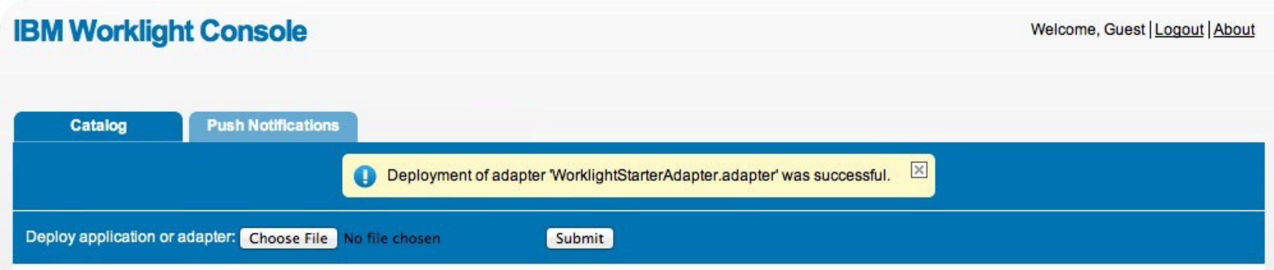
Figure 107. IBM Worklight Catalog

To deploy an adapter:

Procedure

1. Click **Browse**, then navigate to the `.adapter` file and select it.
2. Click Submit.

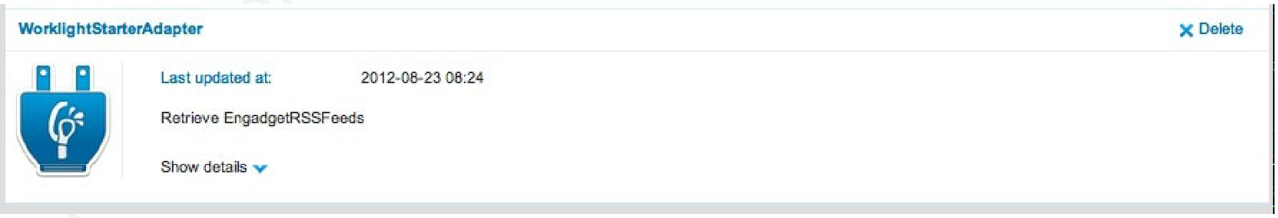
   A message is displayed indicating whether the deployment action succeeded or failed.



Deployment of adapter 'WorklightStarterAdapter.adapter' was successful.

Figure 108. IBM Worklight adapter deployment success or failure message

As a result, the details of the deployed adapter are added to the catalog:

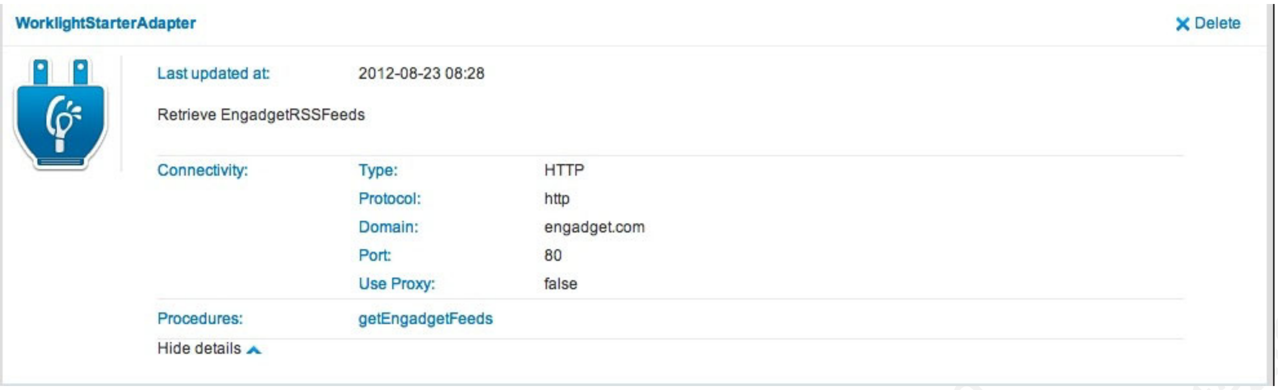


Last updated at: 2012-08-23 08:24

Retrieve EngadgetRSSFeeds

Figure 109. Deployed adapter details

3. Click **Show details** to view connectivity details for the adapter and the list of procedures it exposes.


WorklightStarterAdapter

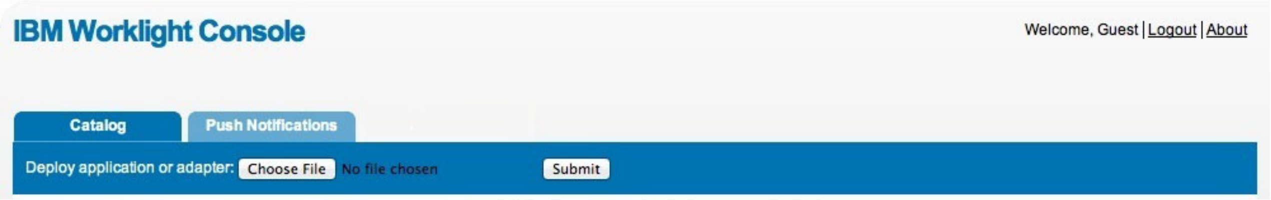
Last updated at:	2012-08-23 08:28	
Retrieve EngadgetRSSFeeds		
Connectivity:	Type:	HTTP
	Protocol:	http
	Domain:	engadget.com
	Port:	80
	Use Proxy:	false
Procedures:	getEngadgetFeeds	

Figure 110. Adapter connectivity details

4. Repeat steps 1 – 3 for each adapter.

   To deploy an application:

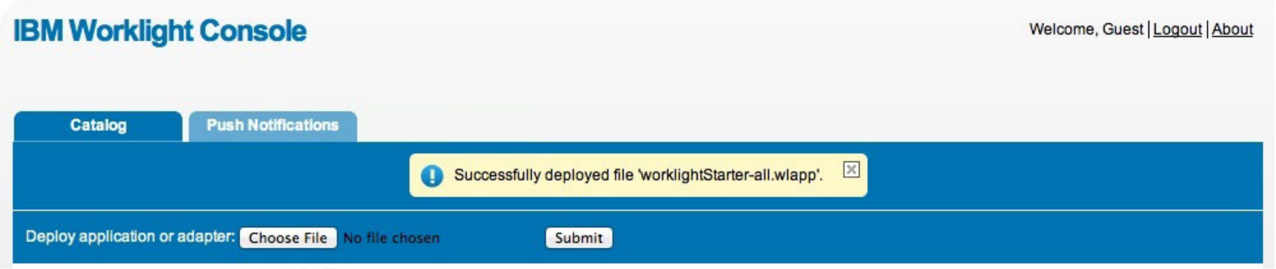
5. In the catalog page, click **Browse**, then navigate to the `.wlapp` file and select it.


IBM Worklight Console

Deploy application or adapter: No file chosen

Figure 111. Browsing the catalog page to find the `.wlapp` file

6. Click **Submit**. A message is displayed indicating whether the deployment action succeeded or failed.

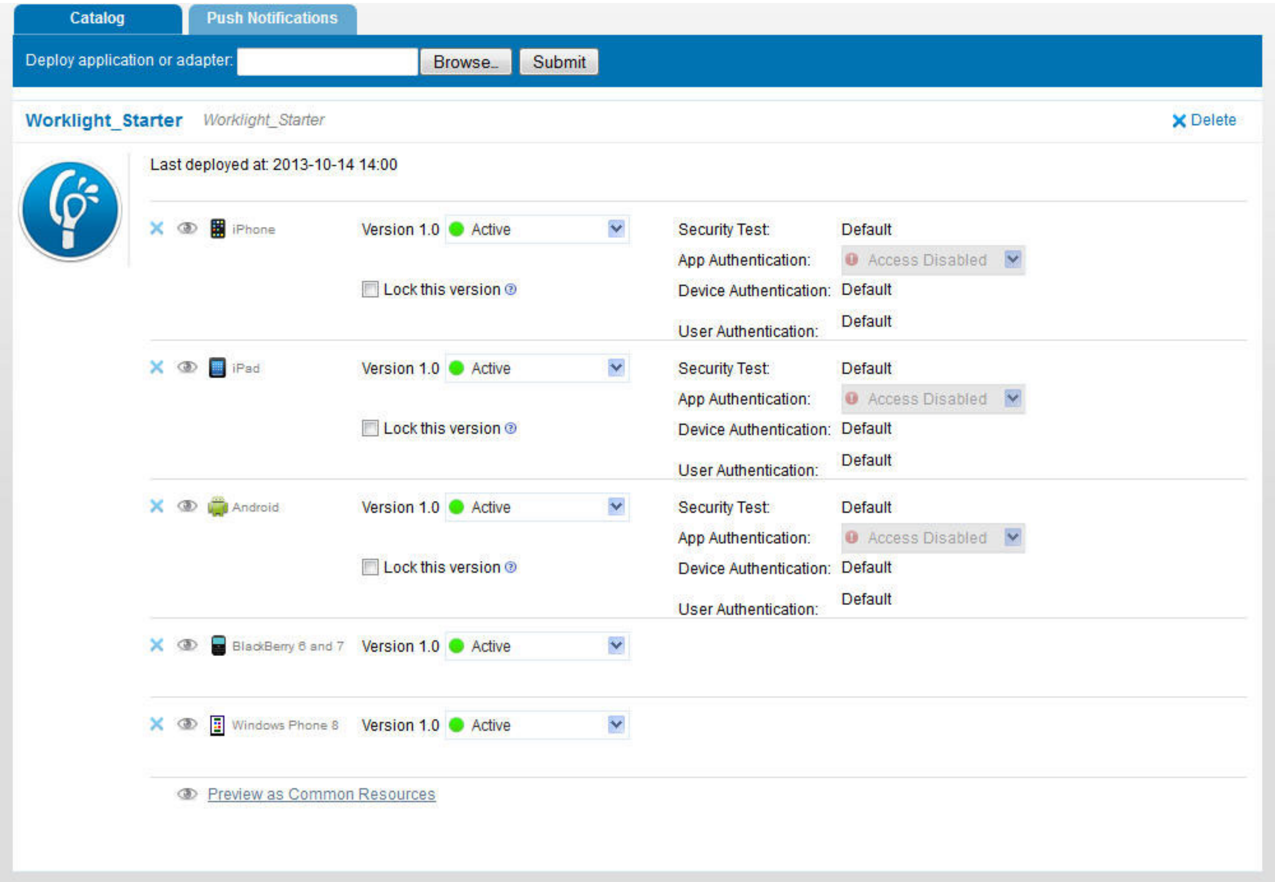

IBM Worklight Console

Successfully deployed file 'worklightStarter-all.wlapp'.

Deploy application or adapter: No file chosen

Figure 112. Deployment success or failure message

As a result, the details of the deployed application are added to the catalog.



Worklight_Starter *Worklight_Starter*

Last deployed at: 2013-10-14 14:00

iPhone Version 1.0 ● Active
□ Lock this version
Security Test: Default
App Authentication: Access Disabled
Device Authentication: Default
User Authentication: Default

iPad Version 1.0 ● Active
□ Lock this version
Security Test: Default
App Authentication: Access Disabled
Device Authentication: Default
User Authentication: Default

Android Version 1.0 ● Active
□ Lock this version
Security Test: Default
App Authentication: Access Disabled
Device Authentication: Default
User Authentication: Default

BlackBerry 6 and 7 Version 1.0 ● Active

Windows Phone 8 Version 1.0 ● Active

Figure 113. Details of the deployed application

7. Repeat steps 1 and 2 for each app.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

Administering adapters and apps in Worklight Console

Open the console before performing administrative tasks.

About this task

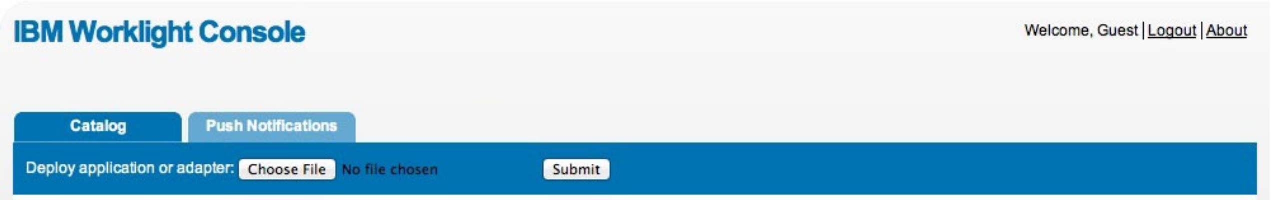
Before performing any of the other tasks in this collection of topics, open the console:

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 Catalog page of the IBM Worklight Console 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

To delete an app, click **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

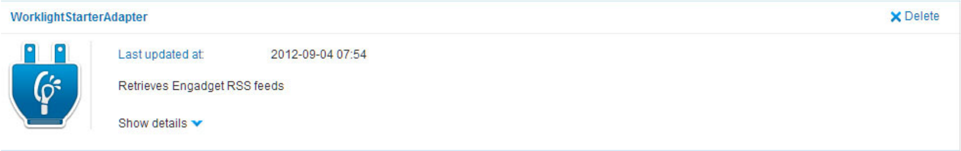
Deploy an adapter from the console.

Procedure

To deploy an adapter:

1. Click **Browse**, then navigate to your `.adapter` file and select it.

2. Click **Submit**.



WorklightStarterAdapter ✕ Delete

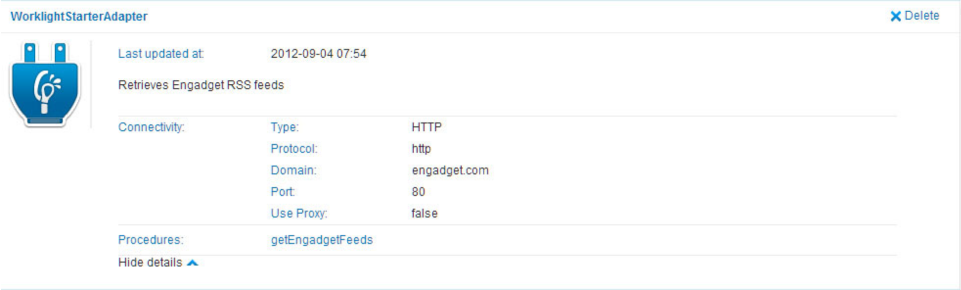
Last updated at: 2012-09-04 07:54

Retrieves Engadget RSS feeds

Show details ⌄

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.



WorklightStarterAdapter ✕ Delete

Last updated at: 2012-09-04 07:54

Retrieves Engadget RSS feeds

Connectivity:	Type:	HTTP
	Protocol:	http
	Domain:	engadget.com
	Port:	80
	Use Proxy:	false
Procedures:	getEngadgetFeeds	

Hide details ⌃

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

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” on page 799.

Results

The new adapter replaces the original one.

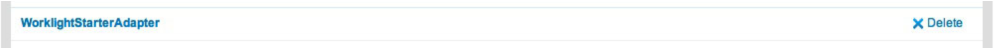
Deleting adapters

Delete an adapter by clicking **Delete**.

Procedure

To delete an adapter:

Click **Delete** to the right of the adapter name.



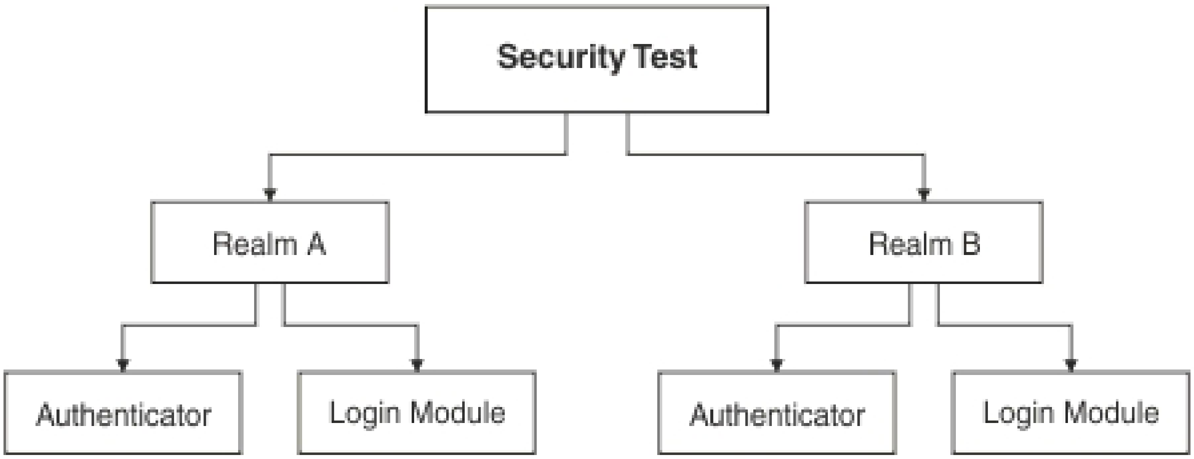
WorklightStarterAdapter ✕ Delete

Worklight Security and LTPA overview

Worklight has comprehensive support for various authentication and authorization methods, including Lightweight Third-Party Authentication (LTPA).

Worklight security basics

The following image shows the authentication elements hierarchy:



Security test
A security test is a set of tests that are used to protect a resource, such as an adapter procedure or application environment. A test includes information about which realm is required to authenticate and other parameters, such as authentication order. A protected resource is accessible only after the client authenticates to all of the tests that are specified in the security test. If the client is unable to log in to all tests, the request to access the protected resource is denied. Individual adapter procedures or an entire application environment can be protected by a security test. For more information about security tests and the different types of security tests, see "Security Tests" on page 605.

Realm A realm creates a relationship between a Worklight login module and a Worklight authenticator to provide a means of authentication. For more information about realms, see "Authentication realms" on page 607.

Authenticator
An authenticator parses incoming requests from a Worklight client to search for required credentials when a protected resource is requested. If credentials are not available in the request, the authenticator is responsible for challenging the client to authenticate. The credentials, after received correctly from the client, are formatted to the login module's predefined requirements and sent to the login module. For more information about authenticators, see "Authenticators and Login Modules" on page 608.

Login module
After an authenticator is able to parse credentials from a request, they are sent to a login module that is responsible for validating those credentials. After the credentials are considered valid and the user can be authorized, the login module creates a user identity for the realm. For more information about login modules, see "Authenticators and Login Modules" on page 608.

User identity

After a login module successfully validates a set of user credentials, it creates a user identity. A user identity contains at least a user name and a display name. It can also contain attributes that provide more details the protected resource might need.

Challenge handlers

A challenge handler is the client-side JavaScript that is included into a Worklight application that is created by the developer. A challenge handler handles an authentication challenge from the server. A challenge handler can be defined for each realm, and is responsible for the following tasks:

- Determine whether a request is an authentication challenge that is specific to the realm.
- Perform necessary user interaction if it receives a challenge.
- Send the credentials to the server to complete the authentication.
- Validate that the authentication was successful.

Worklight security configuration

For a Worklight Server to protect a resource, such as an adapter procedure or an application environment, the administrator must first configure the Worklight Server.

Defining a login module

A login module is the most basic security element in the Worklight authentication configuration. You can define a login module in the `<loginModules>` element in the `authenticationConfig.xml` file. The following example shows a login module definition:

```
<loginModules>
...
  <loginModule name="HeaderLogin"
               canBeResourceLogin="true"
               isIdentityAssociationKey="true"
               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>
...
</loginModules>
```

In this example, the login module is called `HeaderLogin`, and is referred to from a realm element. The `<className>` element must contain the full Java namespace to a login module implementation. The `HeaderLoginModule` is a login module that is included by default. It checks to make sure that the user entered any, non-empty user name and password.

Defining a realm

After a login module is defined, you must specify a realm. You can add a realm to the `<realms>` element in the `authenticationConfig.xml` file. The following example shows a realm definition:

```
<realms>
...
  <realm name="RequiresUserHeaders" loginModule="HeaderLogin">
```

```
    <className>com.worklight.core.auth.ext.HeaderAuthenticator</className>
  </realm>
...
</realms>
```

In this example, the realm is called RequiresUserHeaders, and uses the login module that was defined in the previous example. The <className> element must contain the full Java namespace to an authentication implementation. This realm definition creates a link between an authenticator and a login module.

Defining a security test

A security test can be defined in the <securityTests> element in the authenticationConfig.xml file. The following example shows a security test definition:

```
<securityTests>
...
  <customSecurityTest name="BasicRequirements">
    <test realm="wl_antiXSRFRealm" />
    <test realm="wl_authenticityRealm" />
    <test realm="wl_remoteDisableRealm" />
    <test realm="RequiresUserHeaders" isInternalUserID="true" />
    <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" />
  </customSecurityTest>
...
</securityTests>
```

This custom security test is called BasicRequirements, and contains a list of tests. The tests define which realms are required for authorization into the protected resource. The tests in this example are built in realms. Built in realms are prefixed with wl_.

Note: If one test fails, then the entire security test fails.

The isInternalUserID attributes can be set to true only on a single realm. This attribute is used as the default identity for a user in the security test. The isInternalDeviceID attribute is similar, but sets a default device identity.

This example uses the RequiresUserHeaders realm in the previous example.

Creating a challenge handler

You must create a challenge handler for your Worklight app to handle any custom challenges.For more information about challenge handlers, see the module *Custom Authenticator and Login Module* under category 8, *Authentication and security*, in Chapter 3, “Tutorials and samples,” on page 27.

Worklight application environment protection

After a security test is configured with the appropriate realms, you can protect any resource. One option is to protect an application’s environment, such as iPhone, iPad, or Android, completely with that security test.

To set up this protection, you must add the securityTest attribute to the environment’s element in the applicationDescriptor.xml file. The following example shows the environment protection definition:

```
<android version="1.0" securityTest="BasicRequirements">
...
</android>
```

This definition requires every Android device that connects to the server through your application to log in to the `BasicRequirements` security test.

Worklight adapter procedure protection

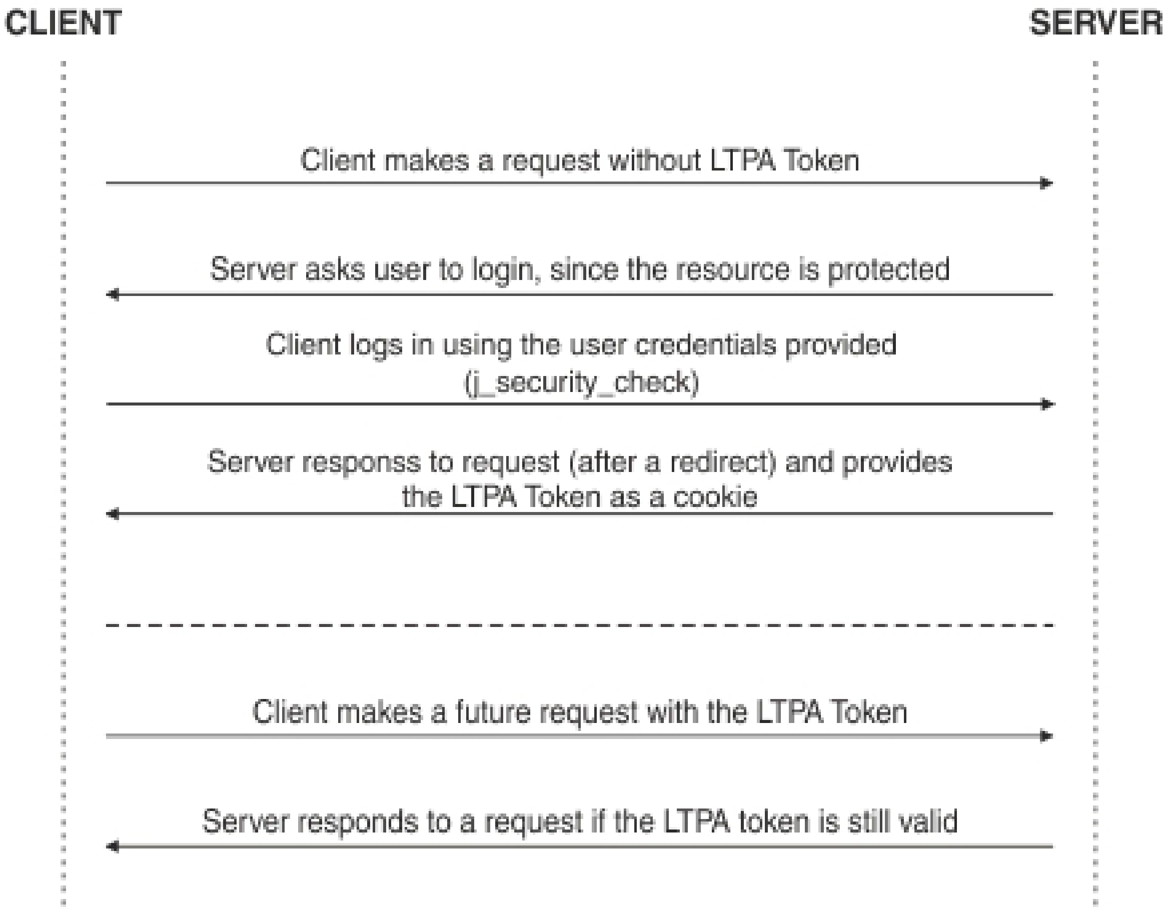
Another option is to protect a Worklight adapter procedure. Using the same security test, you can protect an adapter procedure. When the procedure is called and the user is not already authenticated into the security test, the client is required to authenticate. If you have an adapter procedure named `GetSecretData`, you can protect it in the adapter's XML configuration file by adding the `securityTest` attribute to the `<procedure>` element:

```
<procedure name="GetSecretData" securityTest="BasicRequirements" />
```

Worklight Security and LTPA

Lightweight Third-Party Authentication (LTPA) is a security token type that is used by IBM WebSphere Application Server and other IBM products. LTPA can be used to send the credentials of an authenticated user to backend services. It can also be used as a single sign-on (SSO) token between the user and multiple servers.

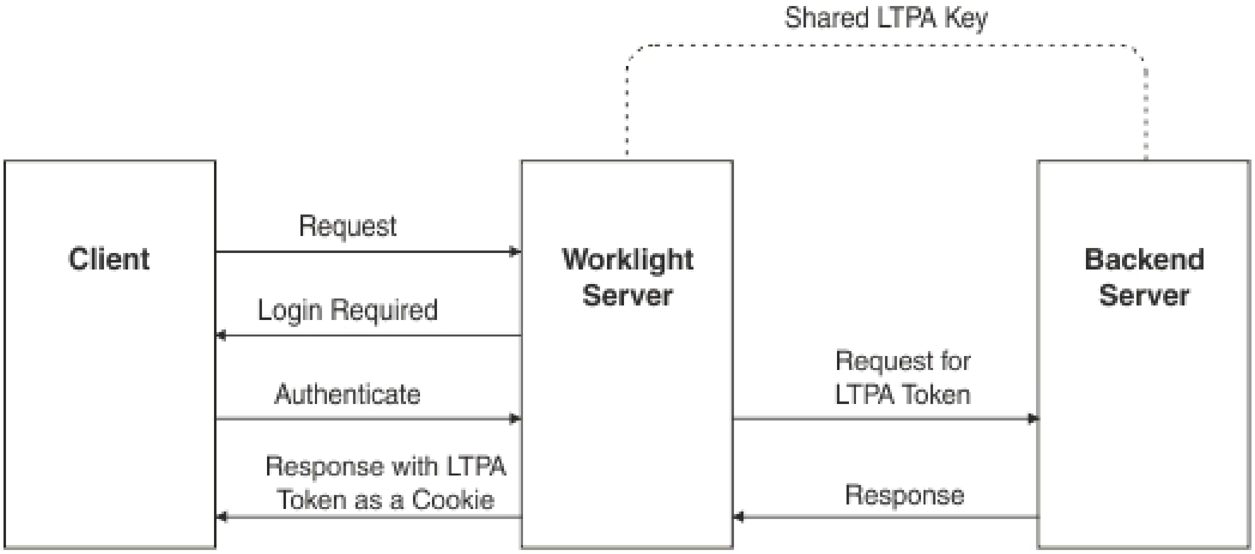
The following image shows a simple client/server flow with LTPA:


CLIENT
SERVER
Client makes a request without LTPA Token
Server asks user to login, since the resource is protected
Client logs in using the user credentials provided (j_security_check)
Server responss to request (after a redirect) and provides the LTPA Token as a cookie
Client makes a future request with the LTPA Token
Server responds to a request if the LTPA token is still valid

After a user logs in, the server generates an LTPA token, which is an encrypted hash that contains authenticated user information. The token is signed by a private key that is shared among all the servers that want to decode it. The token is usually in cookie form for HTTP services. By sending the token as a cookie, there is no need for subsequent user interaction.

LTPA tokens have a configurable expiration time to reduce the possibility for session hijacking.

The following image shows a client-server-backend flow with LTPA:



Your infrastructure can also use the LTPA token to communicate with a backend server to act on behalf of the user. The user cannot directly access the backend server. Enterprise environments should use a reverse proxy, such as DataPower or IBM ISAM, in the DMZ, and place the Worklight Server in the intranet. This configuration ensures that access to the Worklight Server cannot be obtained until a user authenticates. For more information, see "Reverse proxy with LTPA" on page 810.

Configuring the Worklight LTPA realm:

The Worklight Server contains the authenticator and login module that are designed to handle authentication by using LTPA through form-base authentication.

About this task

You must update the `authenticationConfig.xml` file to configure your server to use the Worklight LTPA realm.

Procedure

1. Add the login module definition to the `<loginModules>` element in your server's `authenticationConfig.xml` file. The following example uses a login module that is called `WASLTPAModule`:

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

2. Add the realm definition to the `<realms>` element in your server's `authenticationConfig.xml` file. The following example uses a realm that is called `WASLTPARealm`:

```
<realms>
...
  <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>
```

3. Add a user test to an existing test in the `authenticationConfig.xml` file.

```
<customSecurityTest name="LTPASecurityTest">
  <test realm="wl_authenticityRealm" />
  <test realm="WASLTPARealm" isInternalUserID="true" />
  <test realm="wl_deviceNoProvisiongRealm" isInternalDeviceID="true" />
</customSecurityTest>
```

4. Create a login page and a login error page. The `WASLTPARealm` must know which HTML file to present to the client when the client must authenticate. This HTML file must be named `login.html`. When the client enters invalid credentials, the `WASLTPARealm` presents an error HTML file. This HTML file must be named `loginError.html`. These HTML files must be added to the root directory in the Worklight Server WAR file. The following example shows a sample `login.html` file:

```
<html>
  <head>
    <title>Login</title>
  </head>
  <body>
    <form method="post" action="j_security_check">
      <input type="text"
             id="j_username"
             name="j_username"
             placeholder="User name" />
      <input type="password"
             id="j_password"
             name="j_password"
             placeholder="Password" />
      <input type="submit" id="login" name="login" value="Log In" />
    </form>
  </body>
</html>
```

The following example shows a sample `loginError.html` file:

```
<html>
  <head>
    <title>Login Error</title>
  </head>
  <body>
    An error occurred while trying to log in.
  </body>
</html>
```

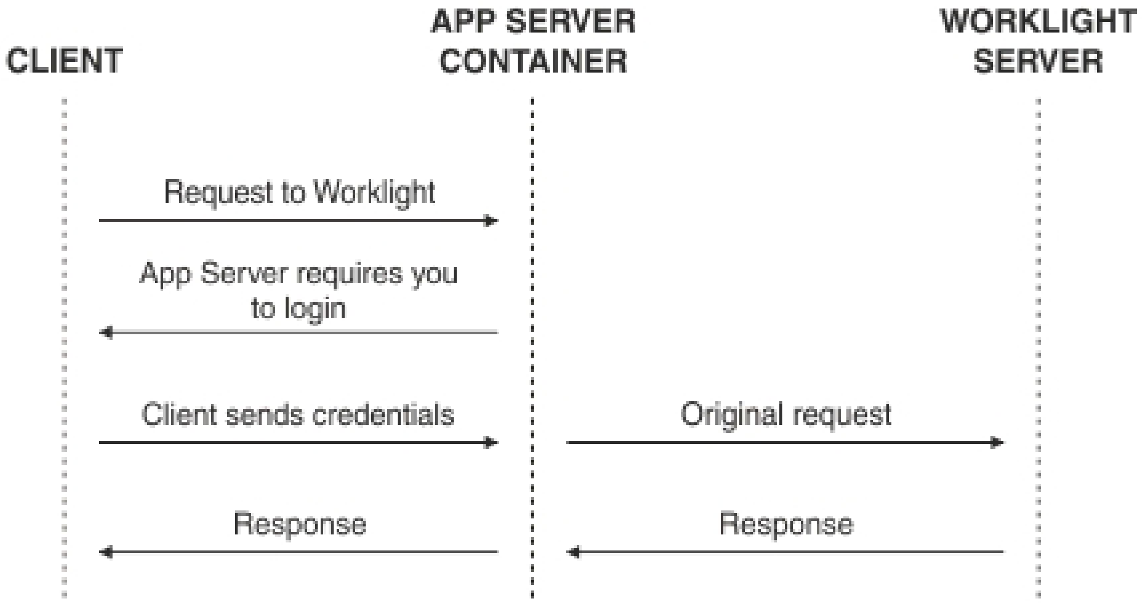
Supported configurations for LTPA

IBM Worklight supports different configuration options to take advantage of LTPA, based on the server configuration and administrative requirements.

Protective application server (Option 1)

This configuration is formally known as Option 1 in the module *WebSphere LTPA-based authentication*, under category 8, *Authentication and security*, in Chapter 3, "Tutorials and samples," on page 27. The application server is configured to protect all resources in the Worklight Server application, which is given specified roles. The application server sends the login page if the user does not send a valid LTPA token with the request. After the user sends valid credentials, the original request is sent to the Worklight Server application with an LTPA token. The LTPA realm consumes the LTPA token and automatically logs in the user.

The following image shows a protective application server flow:



This option is not preferred for new configurations. The application server such as the IBM WebSphere Application Server Liberty Profile (Liberty) protects all resources and forces users to log in before any other authentication mechanism. The behavior occurs regardless of the expected authentication order for a security test.

To use this option with Liberty, you must edit the `web.xml` from the Worklight Server WAR file and Liberty's `server.xml` file. The following example shows the required modifications to the `web.xml` file:

```
<!-- Existing web.xml configuration here -->

<security-constraint id="worklightSecurityConstraint">
  <web-resource-collection id="worklightWebResourceCollection">
    <web-resource-name>Worklight Server</web-resource-name>
    <description>Protection area for Worklight Server.</description>
    <url-pattern>/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint id="worklightAuthConstraint">
    <description></description>
    <role-name>allAuthenticationUsers</role-name>
  </auth-constraint>
  <user-data-constraint id="worklightUserDataConstraint">
    <transport-guarantee>NONE</transport-guarantee>
  </user-data-constraint>
</security-constraint>

<security-role id="securityRoleAllAuthenticatedUsers">
  <description>All Authenticated Users Role.</description>
  <role-name>allAuthenticationUsers</role-name>
</security-role>

<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/loginError.html</form-error-page>
  </form-login-config>
</login-config>
```

The following example shows the required modifications to the server.xml file:

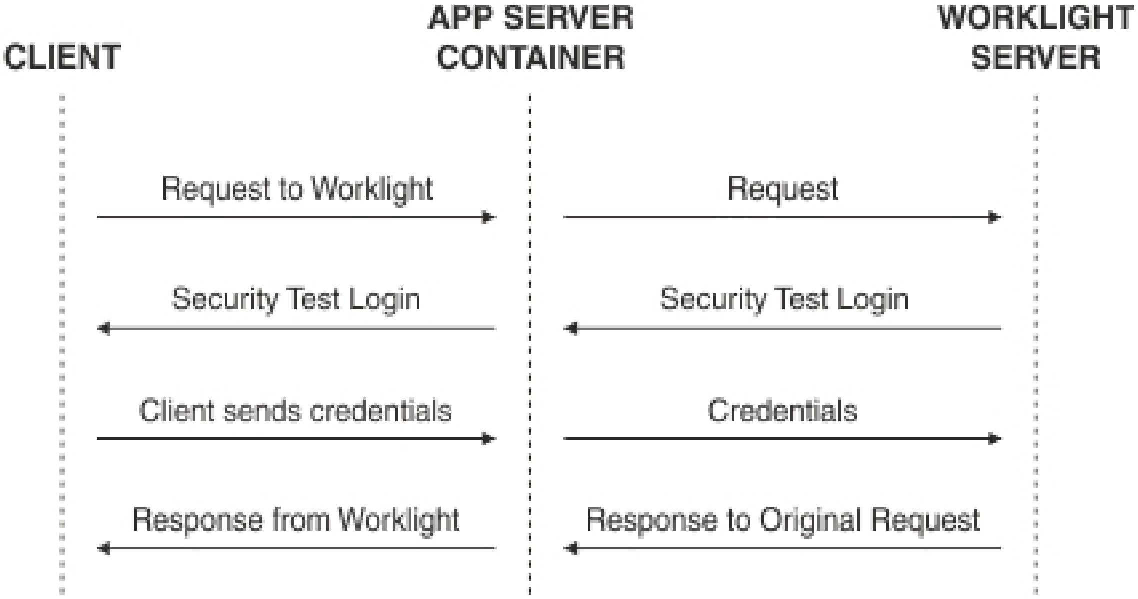
```
<featureManager>
  <feature>appSecurity-1.0</feature>
  <!-- Any additional features you need go here -->
</featureManager>

<!-- Use an existing user registry instead if you have one.
     This registry is just an example. -->
<basicRegistry>
  <user name="sampleuser" password="demo" />
</basicRegistry>

<!-- Any additional settings go here -->
<application context-root="/worklight"
             location="worklight.war"
             name="worklight"
             type="war">
  <classloader commonLibraryRef="worklight-6.1.0,apps-common" />

    <!-- This is our addition: application-bnd.
         The security-role defines who is authorized into a role from web.xml -->
    <application-bnd>
      <security-role name="allAuthenticationUsers">
        <special-subject type="ALL_AUTHENTICATED_USERS" />
      </security-role>
    </application-bnd>
  </classloader>
</application>
```

Note: Remember to add the login.html and loginError.html files to the root directory in the Worklight Server WAR file to provide a way for the user to log in. For more information, see step 4 of “Configuring the Worklight LTPA realm” on page 805.

Protective IBM Worklight security test (Option 2)

An alternative configuration allows the server to use all of the Worklight security test configuration features. This option is preferred for new configurations. For example, Option 1 always asks the user to log in on the first request. Option 2 asks for the user to authenticate only when the Worklight Server deems that it is necessary.

The following image shows a protective security test flow:



You need to modify only Liberty's server.xml file to configure this option. The WASLTPARealm handles the actual authentication against the user registry that is defined in the server.xml file. The example configuration allows the user with the user name sample user and the password demo to authorize correctly.

The following example shows the required modifications to the server.xml file:

```
<featureManager>
  <feature>appSecurity-1.0</feature>
  <!-- Any additional features you need go here -->
</featureManager>

<!-- Use an existing user registry instead if you have one.
     This registry is just an example. -->
<basicRegistry>
  <user name="sampleuser" password="demo"/>
</basicRegistry>

<!-- Any additional settings go here -->
<application context-root="/worklight"
             location="worklight.war"
             name="worklight"
             type="war">
  <classloader commonLibraryRef="worklight-6.1.0,apps-common"/>
    <!-- This is our addition: application-bnd.
         The security-role defines who is authorized into a role from web.xml -->
    <application-bnd>
      <security-role name="allAuthenticationUsers">
        <special-subject type="ALL_AUTHENTICATED_USERS" />
      </security-role>
    </application-bnd>
  </classloader>
</application>
```

Note: Remember to add the login.html and loginError.html files to the root directory in the Worklight Server WAR file to provide a way for the user to log in. For more information, see step 4 of “Configuring the Worklight LTPA realm” on page 805.

Advanced security features

IBM Worklight supports more features that can use LTPA in advanced scenarios, such as user certificate authentication and role-based authentication.

Role-based authentication

In Worklight V6.1 and later, role-based authentication is supported. This feature allows the Worklight LTPA realm to be configured to restrict access to a specific Java Platform, Enterprise Edition role. The realm denies the user if the user is not authorized to the role that is specified. This feature is optional. By not defining a required role in the realm's configuration, all users get an LTPA token and are authorized if credentials are correct.

For more information, see “WASLTPAModule login module” on page 618.

User certificate authentication

In Worklight V6.1 and later, the User Certificate Authentication feature is supported. This form of authentication allows users to authenticate through an X.509 client certificate over SSL. The realm definition includes parameters to configure the authenticator, which includes the concept of a dependent realm. The dependent realm is a realm that is required to be authenticated before the user certificate can be generated. After the user logs in to the dependent realm, the user certificate authenticator uses the user identity to build the certificate signing request (CSR) and certificate.

For more information, see “User certificate authentication” on page 994.

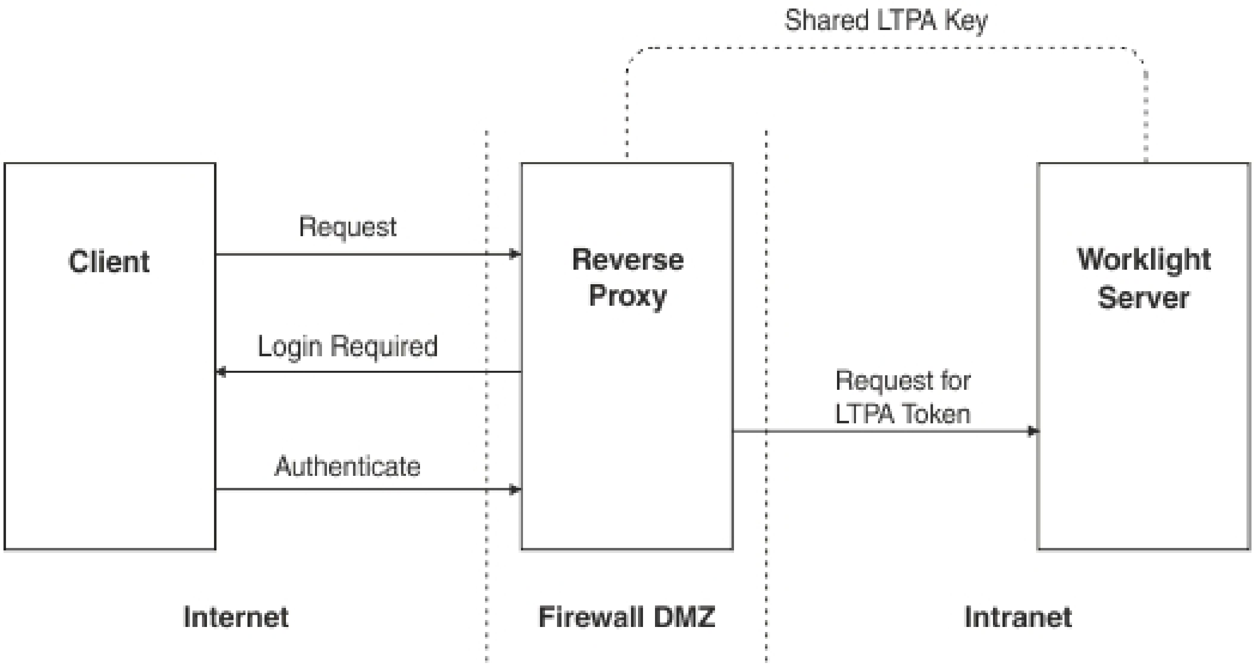
Topologies and use cases

Worklight supports various infrastructure topologies for a set of requirements that can take advantage of LTPA or Worklight security.

Reverse proxy with LTPA

A reverse proxy can be used to authenticate, and then send the user's LTPA token after the user is authenticated. This configuration can be useful when you want to offload Worklight from handling vital user credentials or to use an existing authentication setup. The Worklight Server must be configured for LTPA authentication to get the user identity. Both supported LTPA configurations log the user in automatically if the LTPA token is valid and the user is authorized. For more information about integrating IBM Worklight with a reverse proxy, see “Integration with reverse proxy” on page 1017.

The following image shows a reverse proxy flow:



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 regularly:

Table 165. Data mirroring

Component	Description	Mirror frequency
IBM Worklight Database	All tables must be mirrored. The exceptions to this rule are cache tables (`SSO_LOGIN_CONTEXTS`) and report tables (which are large in size).	Highly dependent on implementation and can range from a few minutes to 24 hours. For more information, 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 back up 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 regularly.

Updating IBM Worklight apps in production

There are general guidelines for upgrading your IBM Worklight apps when they are already in production, on the Application Center or in app stores.

Deploying your IBM Worklight apps for the first time to Worklight Server and the Application Center is covered in other sections of the information center, such as "Deploying an application from development to a test or production environment" on page 711. To recap, the general procedure is as follows:

- Build and test your app in Worklight Studio, and use either the Worklight Console or the supplied Ant tasks to deploy its `.wlapp` file to Worklight Server and the Application Center.
- Submit the generated device app files (such as `.apk` for Android apps and `.ipa` for iOS apps) to their respective app stores (in this example, Google Play and Apple Store).
- Wait for the completion of the review and approval process. Try to avoid updating your app before the review process is completed because doing so can trigger a Direct Update and can confuse the reviewers.

Procedures for upgrading your app when it is already in production are contained in this section. There are several ways to perform such upgrades, depending on their nature:

- Is the upgrade a new version of the app that contains new features or native code, or is it a bug fix or security upgrade?
- Is the upgrade mandatory or optional?
- If it is optional, do you want to leave the old version of the app in place and available to users, or not?
- How and when do you want to notify users of the upgrade?

These subjects are covered in the following topics.

Deploying a new app version and leaving the old version working

The most common upgrade path, used when you introduce new features or modify native code, is to release a new version of your app. Consider following these steps:

1. In Worklight Studio, increment the app version number.
2. Build and test your project and generate new `.wlapp`, `.apk`, and `.ipa` files for it.
3. Deploy the new `.wlapp` file to Worklight Server.
4. Submit the new `.apk` or `.ipa` files to their respective app stores.
5. Wait for review and approval, and for the apps to become available.
6. Optional - send notification message to users of the old version, announcing the new version. See "Displaying a notification message on application startup" on page 840 and "Defining administrator messages from Worklight Console in multiple languages" on page 840.

Deploying a new app version and blocking the old version

This upgrade path is used when you want to force users to upgrade to the new version, and block their access to the old version. Consider following these steps:

1. Optional - send notification message to users of the old version, announcing a mandatory update in a few days. See "Displaying a notification message on application startup" on page 840 and "Defining administrator messages from Worklight Console in multiple languages" on page 840.
2. In Worklight Studio, increment the app version number.
3. Build and test your project and generate new `.wlapp`, `.apk`, and `.ipa` files for it.
4. Deploy the new `.wlapp` file to Worklight Server.

5. Submit the new `.apk` or `.ipa` files to their respective app stores.
6. Wait for review and approval, and for the apps to become available.
7. Copy links to the new app version.
8. Block the old version of the app in Worklight Console, supplying a message and link to the new version. See "Locking an application" on page 836 and "Remotely disabling application connectivity" on page 837.

Note: If you disable the old app, it is no longer able to communicate with Worklight Server. Users can still start the app and work with it offline unless you force a server connection on app startup.

Direct Update (no native code changes)

Direct Update is a mandatory upgrade mechanism that is used to deploy fast fixes to a production app. When you redeploy an app to Worklight Server without changing its version, Worklight Server directly pushes the updated web resources to the device when the user connects to the server. It does not push updated native code. Things to keep in mind when you consider a Direct Update include:

- Direct update does **not** update the app version. The app remains at the same version, but with a different set of web resources. The unchanged version number can introduce confusion if used for the wrong purpose
- Direct update also does not go through the app store review process because it is technically not a new release. This should not be abused because vendors like Apple and Google can become displeased if you deploy a whole new version of your app that bypasses their review. It is your responsibility to read each store's usage agreements and abide by them. Direct update is best used to fix urgent issues that cannot wait for several days.
- Direct Update is considered a security mechanism, and therefore it is mandatory, not optional. When you initiate the Direct Update, all users **must** update their app to be able to use it.
- Direct Update does not work if an application is compiled (built) with a different version of Worklight Studio than the one that was used for the initial deployment.

The steps for initiating a Direct Update are as follows:

1. Optional - send notification message to users of the old version, announcing a mandatory update in the next few hours or days. See "Displaying a notification message on application startup" on page 840 or "Defining administrator messages from Worklight Console in multiple languages" on page 840.
2. In Worklight Studio, do **not** increment the app version number.
3. Build and test your project and generate a new `.wlapp` file for it.
4. Deploy the new `.wlapp` file to Worklight Server. This initiates the Direct Update.

For more information about Direct Update, see "Direct updates of app versions to mobile devices" on page 833.

Deploying to the cloud by using IBM PureApplication System and IBM SmartCloud Orchestrator

IBM Worklight provides the capability to deploy and manage IBM Worklight Servers and applications on IBM PureApplication System and IBM SmartCloud Orchestrator.

Using IBM Worklight in combination with IBM PureApplication System and IBM SmartCloud Orchestrator 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 Mobile Application Platform Pattern Extension for Worklight Studio
Provides Mobile Application Platform Pattern 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 cloud deployment

You must install the IBM Mobile Application Platform Pattern Type and IBM Mobile Application Platform Pattern 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-6.1.0.2.tgz` file in the `worklight_pattern_6.1.0.2.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 “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 **Accept**. 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_6.1.0.2.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 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_6.1.0.2.offering.zip` file.
   a. Click **Browse**, navigate to the folder into which you extracted the `WLRTDB.zip` file, and then select the `WLRTDB.zip` file.
7. Click **Save** 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 819).

Installing IBM Worklight support for cloud deployment from the command line

If you download the IBM PureApplication System command-line interface, you can install IBM Worklight support for cloud deployment by running a Python script from the command line.

Before you begin

You need to download the command-line interface before you run the Python script. For further information about the command-line interface, see http://pic.dhe.ibm.com/infocenter/psappsys/v1r1m0/topic/com.ibm.puresystems.appsys.1500.doc/iwd/cct_usingcli.html.

Procedure

Open a command prompt and run the following Python script:

```
./deployer -h localhost -u cbadmin -p cbadmin -a  -f install.py /dir/mobile

install.py:
import sys
mobileloc=sys.argv[1]

print "Loading Mobile"
deployer.patterntypes.create(mobileloc+'/worklight.ptype-6.1.0.2.tgz')
print "Accept/enable"
deployer.patterntypes.get("worklight.ptype","6.1.0.2").acceptLicense()
deployer.patterntypes.get("worklight.ptype","6.1.0.2").enable()

print "creating db workload standards"
temp=deployer.dbworkloads.create({"rate":"3","workload_type":"Departmental OLTP","initial_disk_siz
deployer.dbworkloads.get(temp['id']).workloadfiles.upload(mobileloc+"/WLRTDB.zip")
temp=deployer.dbworkloads.create({"rate":"3","workload_type":"Departmental OLTP","initial_disk_size"
deployer.dbworkloads.get(temp['id']).workloadfiles.upload(mobileloc+"/WLRPTDB.zip")
```

Installation of IBM Mobile Application Platform Pattern Extension for Worklight Studio

IBM Mobile Application Platform Pattern 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 Mobile Application Platform Pattern Extension is also installed.

For more information about installation and configuration of Worklight Studio, see Chapter 6, “Installing and configuring,” on page 45.

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:

- IBM Worklight application component
- IBM Worklight adapter component
- IBM Worklight configuration component
- IBM Worklight application component link to enterprise application (Websphere Application Server) component
- IBM Worklight adapter component link to enterprise application (Websphere Application Server) component
- Enterprise application (Websphere Application Server) component link to IBM Worklight configuration component
- IBM Worklight configuration link to user registry (Tivoli Directory Server)

Importing the sample IBM Worklight virtual application pattern

You import the sample IBM Worklight virtual application pattern from the IBM PureApplication system console. You need to set the correct database workload standard.

About this task

A sample Worklight application pattern can be found in `worklight_pattern_6.1.0.2.offering.zip` called `WorklightSamplePattern.zip`.

Procedure

1. Import the sample IBM Worklight application pattern.
   a. Log on to the IBM PureApplcation system console.
   b. Select **Patterns** > **Virtual applications**.
   c. In the filter, select **IBM Mobile Application Platform Pattern Type 6.1**.
   d. Click the **Import** icon.
   e. Upload the `WorklightSamplePattern.zip` file.
2. Set the correct database workload standard.
   a. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab to edit the sample Worklight virtual application pattern.
   b. Click **Worklight Runtime DB**, then from the **Source** properties list in the right panel, select **Apply a database workload standard**, and then click the Worklight runtime database workload standard.
   c. Repeat the previous step to set the Worklight reports database workload standard for **Worklight Reports DB**.

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 an optional reports database, and by 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.

About this task

All actions associated with creating and configuring the reports database are optional. The reports database is required only for viewing Worklight reports.

Procedure

1. Create a Worklight Server.
   a. If necessary, use the IBM Mobile Application Platform Pattern Extension or the command line interface to package up the Worklight Server into an EAR file. For more information, see the following topics:
      - “Deploying an IBM Worklight project to IBM PureApplication System or IBM SmartCloud Orchestrator” on page 824
      - “Building an IBM Worklight virtual application” on page 825
   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 166. 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 optional 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 167. 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 **Source** list, select **Apply a database workload standard**, and then click the database workload standard created for the IBM Worklight runtime database (see "Installing custom IBM Worklight database workload standards" on page 816).

   c. Optional: Repeat step 2a to create a reports database.
   d. Optional: Supply the following information in the fields provided to define the reports database:

Table 168. Worklight reports database component properties

Property	Description
Name	Name for the Worklight reports database component; for example, `WL Reports DB`.
Database name	Name for the reports database; for example, `WLRPT`.
Source	From the **Source** list, select **Apply a database workload standard**, and then click the database workload standard created for the IBM Worklight reports database (see "Installing custom IBM Worklight database workload standards" on page 816).

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. Optional: Drag a connection from the Worklight Server component to the reports database component.

Note: A warning message is displayed in the Worklight Server component, indicating “Connection to 'xxx database' is required for res-ref-name 'jdbc/WorklightReportsDS' in module 'xxx.war'”. This warning message can be disregarded.

d. Optional: 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 169. Worklight configuration component properties

Property	Description
Name	Name for the Worklight configuration component.
Worklight Console Protection	Select this check box to enable security protection for the Worklight console. Clear the check box to disable security protection.
Worklight Console Username	User name for Worklight console protection.
Worklight Console Password	Password for Worklight console protection.

5. Create Worklight applications and adapters.

a. From the Assets list, expand **Worklight Components**, and then drag and drop a Worklight adapter component and a Worklight application component onto the canvas.

b. For the Worklight application component, supply the following information in the fields provided:

Table 170. 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 171. 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 825

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 172. Tivoli Directory Server component properties

Property	Description
Name	Name for the directory server.
LDIF file	LDIF file to be uploaded for the Tivoli Directory Server.
Base DN	Effective only when the LDAP login module has the parameter **`baseDN`**.
User filter	Effective only when the LDAP login module has the parameter **`userFilter`**.
Group filter	Effective only when the LDAP login module has the parameter **`groupFilter`**.

Connecting to an existing Tivoli Directory Server:

You connect to an existing Tivoli Directory Server by dragging and dropping a Connect Out component onto the PureApplication System canvas, specifying the IP address and port number of your existing Tivoli Directory Server, and linking the 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 173. Operations on Virtual Application Pattern instances

Operation	Description
Worklight Application/Adapter	Install or update Worklight applications and adapters. Supported file types: `*.wlapp`, `*.adapter`, `*.zip`.
Worklight Console Protection	Enable and disable security protection for the Worklight Console.
Worklight Console Username	Username for Worklight Console protection.
Worklight Console Password	Password for Worklight Console protection.

5. To submit the changes you have made, click **Submit**.
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 Mobile Application Platform Pattern Extension for Worklight Studio

Working with IBM Mobile Application Platform Pattern Extension for Worklight Studio involves setting up cloud environment preferences, deploying your Worklight project to the cloud, and integrating with Tivoli Directory Server.

Note: IBM Mobile Application Platform Pattern Extension for Worklight Studio copies the PureApplication configuration into the `WorklightServerConfig` folder under your eclipse workspace.

Note: Deployment to the cloud from Worklight Studio is for the development environment only. For information about deploying to the cloud in a production environment, see "Working with the IBM Mobile Application Platform Pattern Type" on page 818.

Note: IBM Mobile Application Platform Pattern Extension for Worklight Studio uses several REST APIs to interact with IBM PureApplication System and IBM SmartCloud Orchestrator. Ensure your account has permission to access the following locations:

- `<system host>/resources/environmentProfiles`
- `<system host>/resources/clouds`
- `<system host>/resources/hypervisors`
- `<system host>/resources/databaseWorkloads`

Specifying cloud environment preferences in Worklight Studio

Specify cloud environment preferences in Worklight Studio before you deploy IBM Worklight projects to the cloud.

Procedure

1. In Eclipse, click **Windows** > **Preferences** > **Worklight** > **IBM Mobile Application Platform Pattern**.
2. Supply the following information in the fields provided:

Table 174. Cloud environment preferences

Property	Description
System Host	IP address of the system host.
User name	Account user name for accessing the system host.
Password	Password for accessing the system host.

3. Click **Fetch Deployment Information**. 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 IBM PureApplication System or IBM SmartCloud Orchestrator

You deploy a Worklight project to PureApplication System or SmartCloud Orchestrator by running the project in Eclipse.

Before you begin

Before deploying, 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 773.

Procedure

1. In Eclipse, navigate to the Project Explorer view.
2. Right-click your Worklight project, and then click **Run As** > **Deploy project as IBM Mobile Application Platform Pattern**.

3. Select Worklight applications and adapters to be deployed on PureApplication System or SmartCloud Orchestrator, and then click **Run**.
4. In the Worklight Console, check the status and wait for the project to be deployed. When the project has been deployed, a window opens displaying the Worklight Console URL.

Displaying the Worklight Console URL for a deployed IBM Worklight project

You can display 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 **IBM Mobile Application Platform Pattern** > **Display Worklight Console URL**. A window opens displaying the Worklight Console URL.

Integration with Tivoli Directory Server

To use Tivoli Directory Server as a user registry for your IBM Worklight application on PureApplication System or IBM SmartCloud Orchestrator, 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 or IBM SmartCloud Orchestrator.

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_6.1.0.2.offering.zip` file. Make sure you extract it before you build and deploy Worklight virtual applications with the command line interface.

About this task

The Ant task for building a Worklight virtual application has the following structure:

```
<taskdef resource="com/worklight/ant/defaults.properties"
         classpath="${taskdefClasspath}"/>
<target name="buildIPAS_VAP"
        depends="buildAll" >
    <vap-builder
        worklightWar="${worklightWar}"
        destinationFolder="${wlProjectDestDir}"
        artifactsFolder="${artifactsFolder}"
        elbHost="${elbHost}"/>
</target>
```

The following table describes the attributes.

Table 175. Ant task build attributes

Attributes	Description
worklightWar	Required. The Worklight Console WAR file including the full file path.
destinationFolder	Optional. Default value: `${projectfolder}/bin`.
artifactsFolder	Optional. Folder for adapters and applications.
elbHost	Optional. Host name for elastic load balancer.
createVAPFlag	Optional. Whether to generate a VAP `.zip` file. Default value: `true`.
isConnectNewTDS	Optional. Whether to connect a new Tivoli Directory Service.
ldifFile	Optional. When **isConnectNewTDS** is `true`, you must set this attribute.
ipasModel	Optional. Default value is `W1500`; in this case, it works on Intel. You can set its value to `W1700`; in this case, IBM PureApplication System or IBM SmartCloud Orchestrator runs on Power® system.
ipasHost	Optional. The URL of IBM PureApplication System or IBM SmartCloud Orchestrator. Required when **createVAPFlag** is `true`.
username	Optional. The user name that is required to access the IBM PureApplication System or IBM SmartCloud Orchestrator console. Required when **createVAPFlag** is `true`.
password	Optional. The password that is required to access the IBM PureApplication System or IBM SmartCloud Orchestrator console. Required when **createVAPFlag** 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_6.1.0.2.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 176. Ant task deployment attributes

Attributes	Description
vapZipFile	Required. Path to the zip file built by vap-builder.
ipasHost	Required. URL of IBM PureApplication System.
username	Required. Username to access PureApplication System console.
password	Required. Password to access PureApplication System console.
deploymentTarget	Optional. Deployment target type that is used to deploy VAP. The value can be either “environment profile” (default value) or “cloud group”.
profileName	Required when deploymentTarget is equal to “environment profile”. Profile name for deploying VAP.
cloudGroupName	Required. Cloud group name for deploying VAP.
ipGroupName	Required when deploymentTarget is equal to “environment profile”. IP group name for deploying VAP.
priority	Required when deploymentTarget is equal to “environment profile”. Priority for deploying VAP.
IPVersion	Required when deploymentTarget is equal to “cloud group”. IP version that is used to deploy VAP. The value can be either “IPv4” or “IPv6”.

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 or IBM SmartCloud Orchestrator.

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.

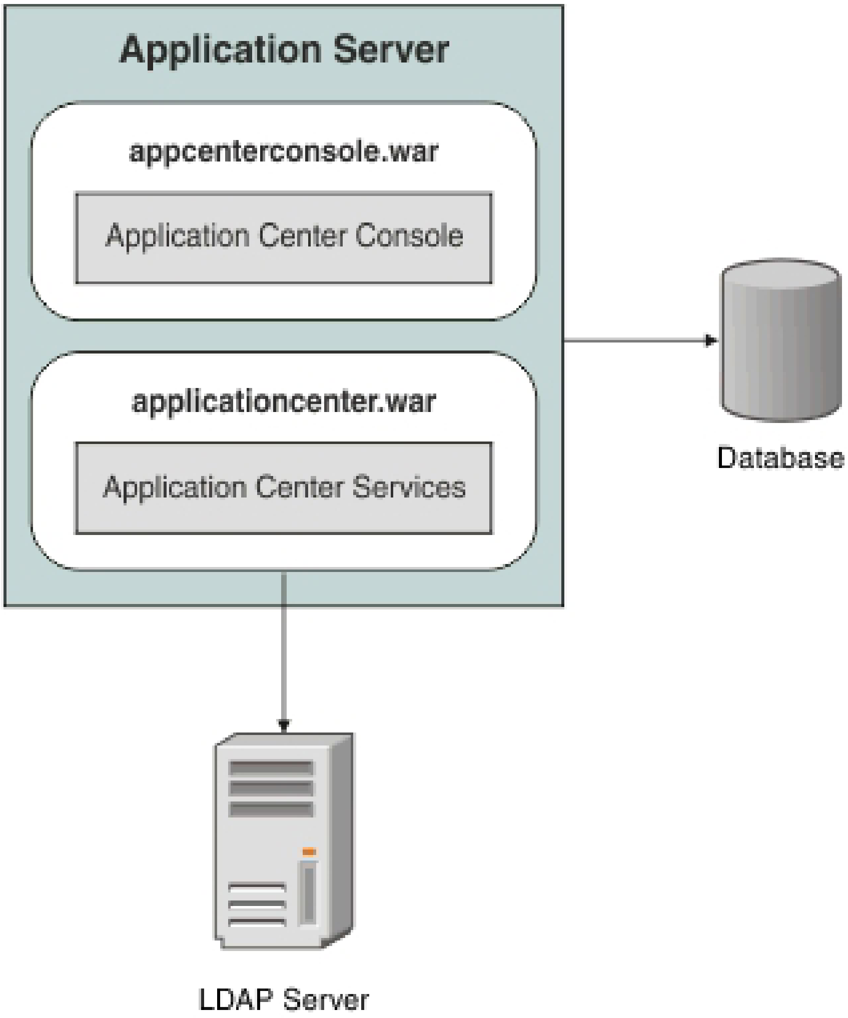


Figure 114. Typical operational model of the Application Center

Related concepts:

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.

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.

Managing users with LDAP
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 52.

Make a note of the path of the installation folder, because later in the procedure you will need some assets that are located in it. When you install the Worklight Server through the IBM Installation Manager, the Application Center artifacts are installed in the `{worklight_install_folder }/ApplicationCenter`.

You must have an IBM PureApplication System environment and the privilege to create Virtual Application Pattern (VAP) and to run Virtual Application instances.

About this task

By following this procedure you prepare the operational components of the Application Center for deployment of the enterprise application on PureApplication System. You connect the operational components to each other and then you can save the configuration and deploy the Application Center on PureApplication System as a web application.

Procedure

1. Get the enterprise archive (EAR) file for the Application Center. This file is located in `{worklight_install_folder }/ApplicationCenter/console`. As of V5.0.6, the Application Center has two web archive (WAR) files, one for the console and one for the services. An EAR file containing them is supplied to simplify deployment on PureApplication System. The context roots of the WAR files within the EAR file are:
   - `/appcenterconsole` for the console
   - `/applicationcenter` for the services

   If you choose to build the EAR file manually, you must remember the context roots of the WAR files.
2. Create the Virtual Application Pattern.
   a. Log in to IBM PureApplication System
   b. Select **Workload Console** > **Patterns** > **Virtual Application Patterns**.
   c. Select **Web Application Pattern Type 2.0**.
   d. Click **+**.
   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.

Chapter 11. 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.

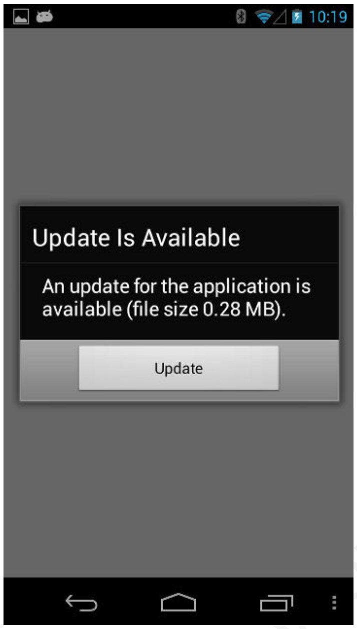
When you redeploy an app to the Worklight Server without changing its version, the Worklight Server directly pushes the web resources (HTML, JavaScript, and CSS) of the newly deployed app to the device. When an app with an older version of these resources connects to the Worklight server, the server 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.

The Direct Update feature works if the server-side artifacts (in this case, the `.wlapp` file) are built with the same version of Worklight Studio used to generate the mobile app. See “Upgrading to Worklight Studio V6.1.0” on page 222 for instructions about how to reenable direct update after an upgrade of Worklight Server.

When the app connects to the Worklight Server, it starts downloading the newly deployed resources, as shown in the following figures. If the download fails mid-way, the direct update will resume from where the download was broken the previous time.

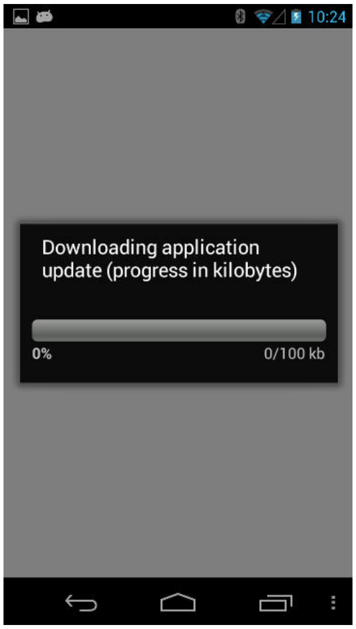
Note: The user notifications seen in Figures 1 through 4 show the default method of implementing Direct Update. Another option is to set the `updateSilently` property of the WL.Client.init method to true, as defined in the WL.Client class. When this is done, the Direct Update is performed silently, without notifying the user before downloading new application resources.



Update Is Available

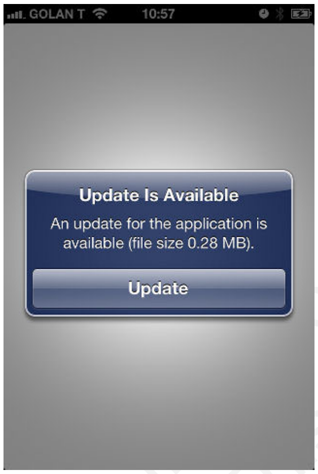
An update for the application is available (file size 0.28 MB).

Figure 115. Update notice from Android



Downloading application update (progress in kilobytes)

Figure 116. Downloading newly deployed resources to Android



Update Is Available

An update for the application is available (file size 0.28 MB).

Figure 117. Update notice from iOS

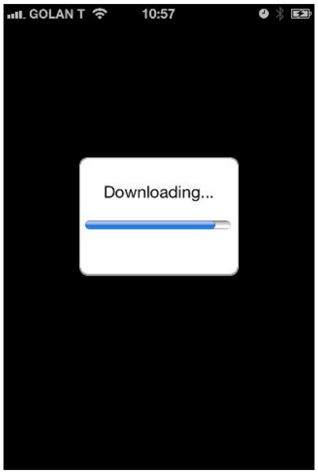


Figure 118. Downloading newly deployed resources to iOS

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.

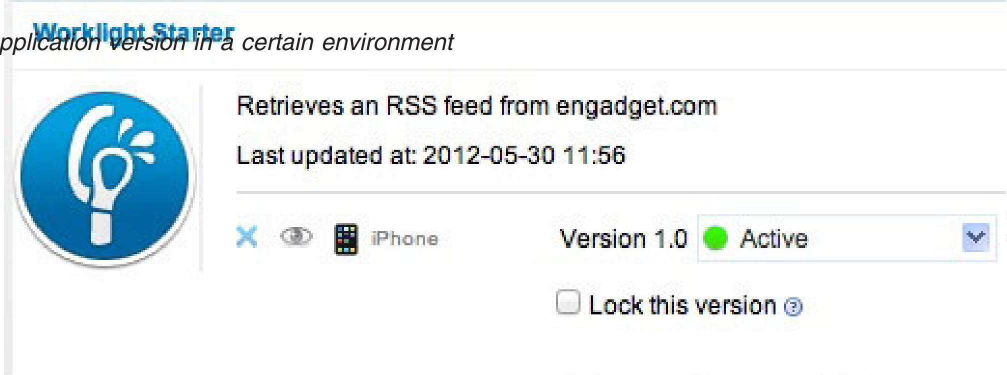
About this task

You can lock applications for iPhone, iPad, and Android.

Procedure

To lock an application version in a certain environment, select the **Lock this version** check box for that application version in the relevant environment.

Figure 119. Locking an application version in a certain environment



Worklight Starter

Retrieves an RSS feed from engadget.com

Last updated at: 2012-05-30 11:56

iPhone

Version 1.0 ● Active

□ Lock this version

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.

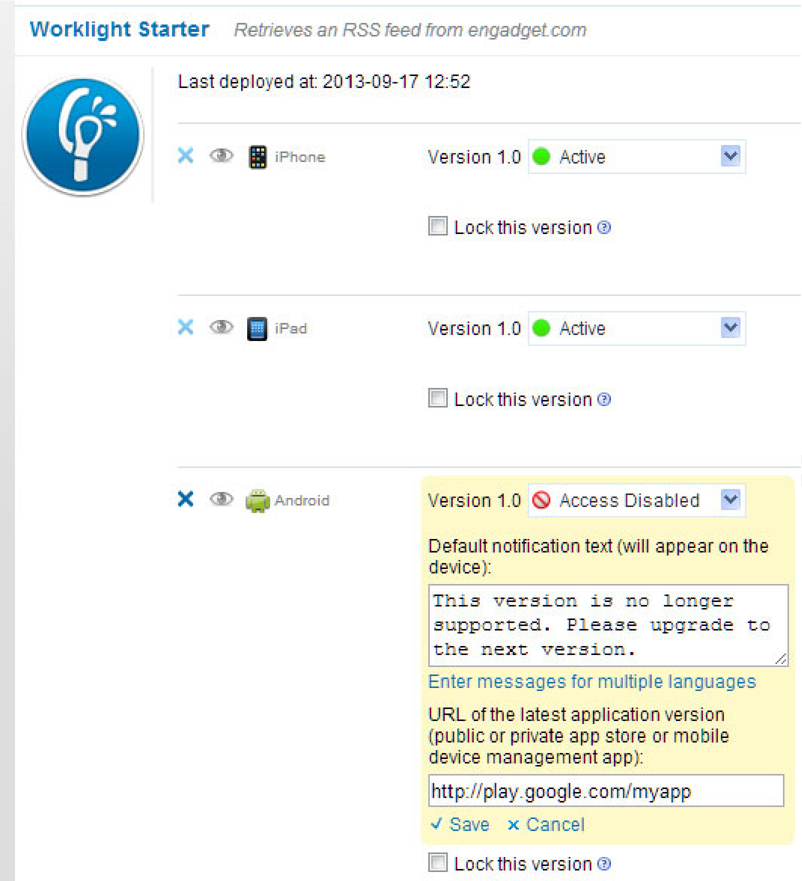
About this task

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

1. To use this Remote Disable feature, change the status of the application version that must be disabled from **Active** to **Access Disabled**.
2. Add a custom message as shown in the following figure:

Figure 120. Denying access to older application versions



Worklight Starter *Retrieves an RSS feed from engadget.com*

Last deployed at: 2013-09-17 12:52

iPhone Version 1.0 ● Active

□ Lock this version

iPad Version 1.0 ● Active

□ Lock this version

Android Version 1.0 Access Disabled

Default notification text (will appear on the device):

```
This version is no longer
supported. Please upgrade to
the next version.
```

Enter messages for multiple languages

URL of the latest application version (public or private app store or mobile device management app):

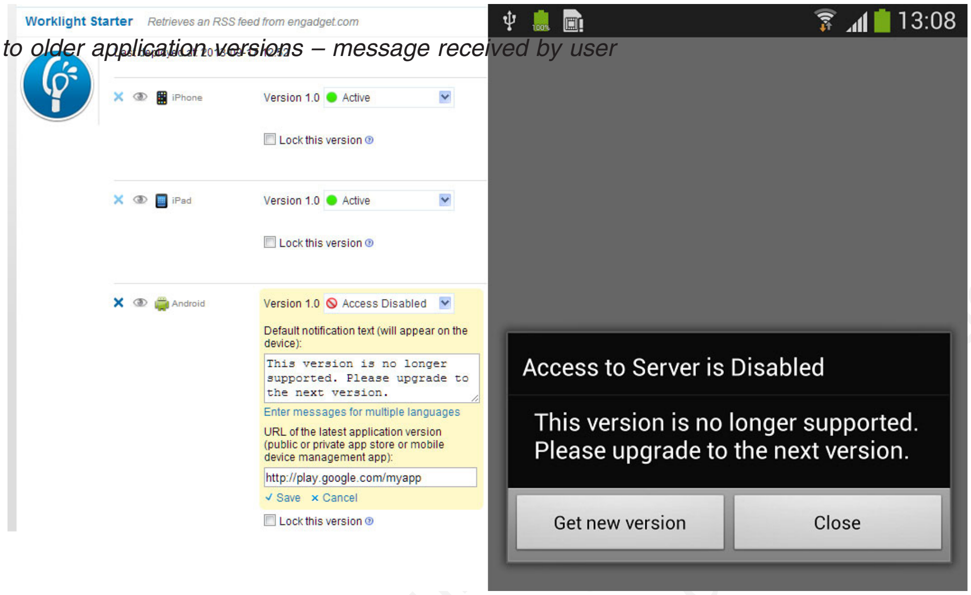
http://play.google.com/myapp

✓ Save × Cancel

□ Lock this version

You can also specify a URL for the new version of the application (usually in the appropriate app store or market).

When users run an application that is Remotely Disabled, they receive a text message about the access denial. They can either close the dialog and continue working offline (that is, without access to the Worklight Server), or they can upgrade to the latest version of the application. Closing the dialog keeps the application running, but any application interaction that requires server connectivity causes the dialog to be displayed again.



Worklight Starter Retrieves an RSS feed from engadget.com

iPhone Version 1.0 ● Active

□ Lock this version

iPad Version 1.0 ● Active

□ Lock this version

Android Version 1.0 Access Disabled

Default notification text (will appear on the device):

This version is no longer supported. Please upgrade to the next version.

Enter messages for multiple languages

URL of the latest application version (public or private app store or mobile device management app):

http://play.google.com/myapp

✓ Save × Cancel

□ Lock this version

13:08

Access to Server is Disabled

This version is no longer supported. Please upgrade to the next version.

Figure 121. 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 or end 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 840

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

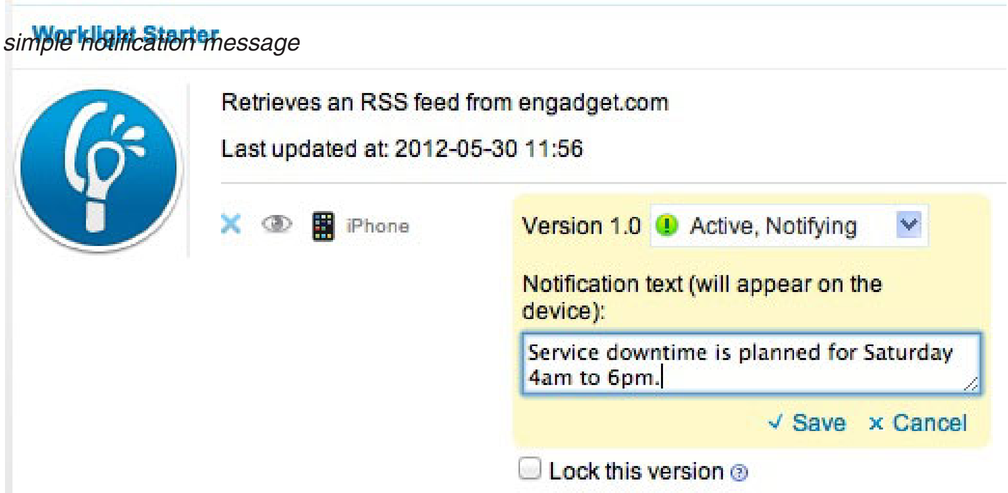
You can set a notification message that is displayed for the user when the application starts, but does not cause the application to exit.

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:



Worklight Starter

Retrieves an RSS feed from engadget.com

Last updated at: 2012-05-30 11:56

Version 1.0 Active, Notifying

Notification text (will appear on the device):

Service downtime is planned for Saturday 4am to 6pm.

Figure 122. 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.

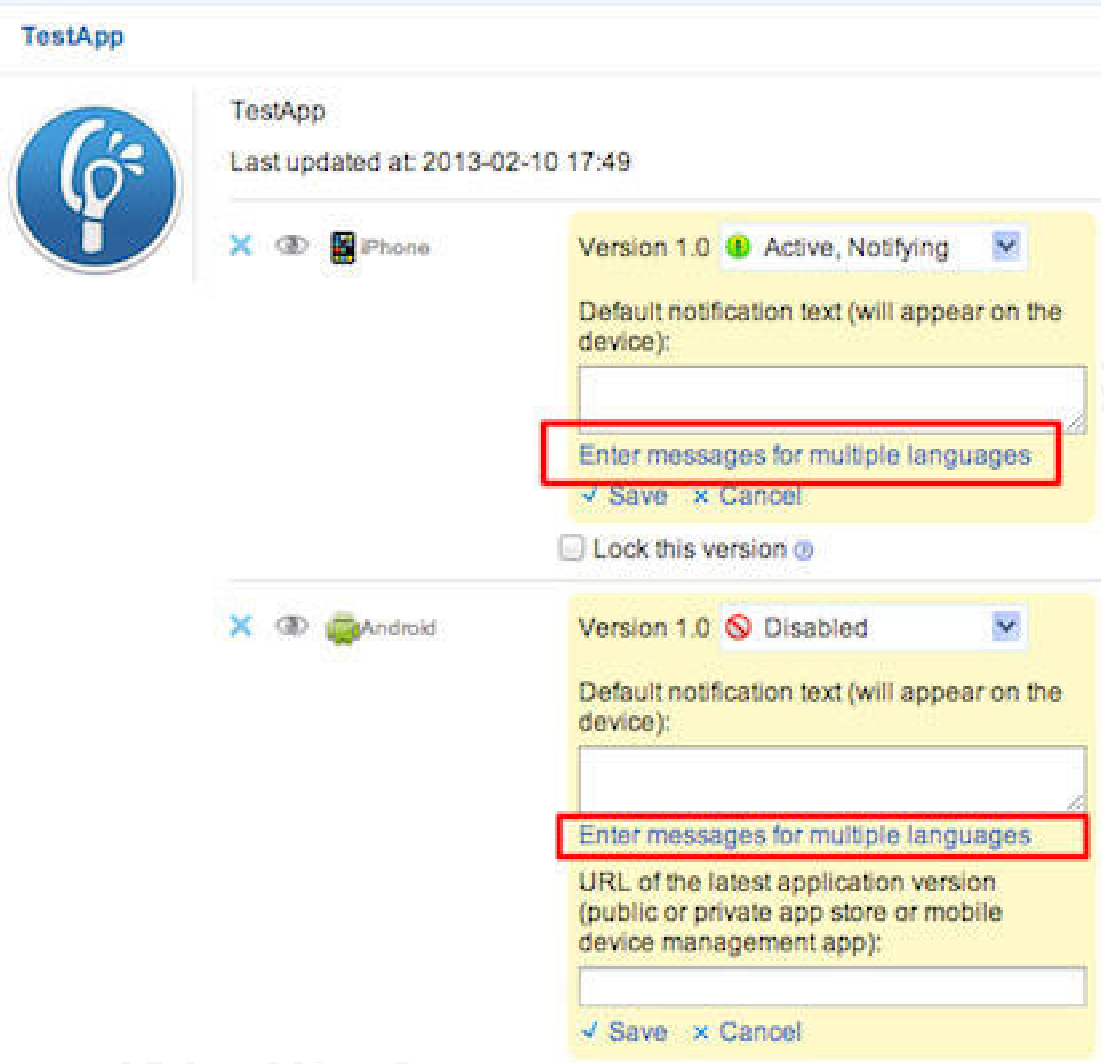
Defining administrator messages from Worklight Console in multiple languages

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**.



TestApp

TestApp

Last updated at: 2013-02-10 17:49

iPhone

Version 1.0 Active, Notifying

Default notification text (will appear on the device):

Enter messages for multiple languages

✓ Save × Cancel

□ Lock this version

Android

Version 1.0 Disabled

Default notification text (will appear on the device):

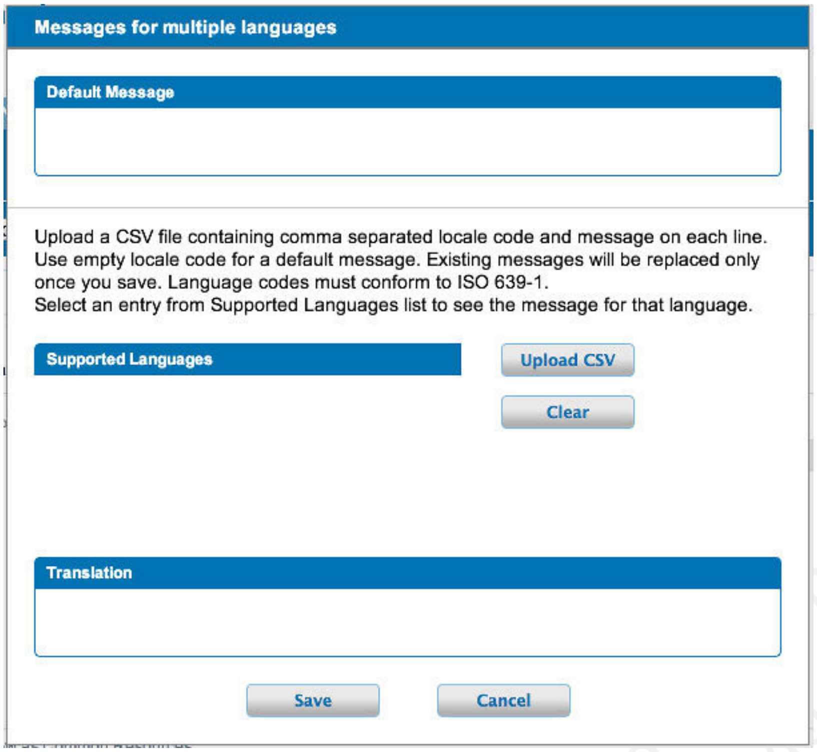
Enter messages for multiple languages

URL of the latest application version (public or private app store or mobile device management app):

✓ Save × Cancel

Figure 123. 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.



Messages for multiple languages

Default Message

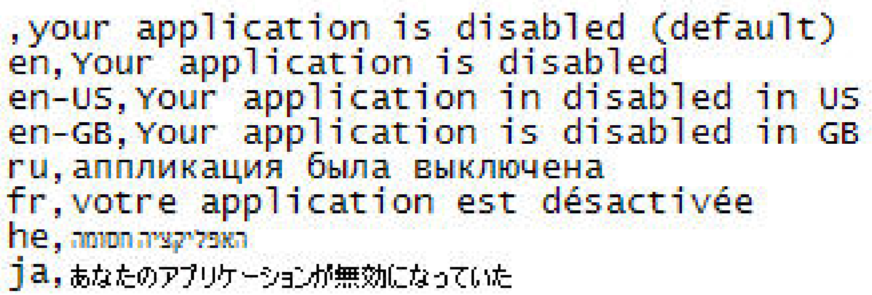
Upload a CSV file containing comma separated locale code and message on each line. Use empty locale code for a default message. Existing messages will be replaced only once you save. Language codes must conform to ISO 639-1.
Select an entry from Supported Languages list to see the message for that language.

Figure 124. 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:

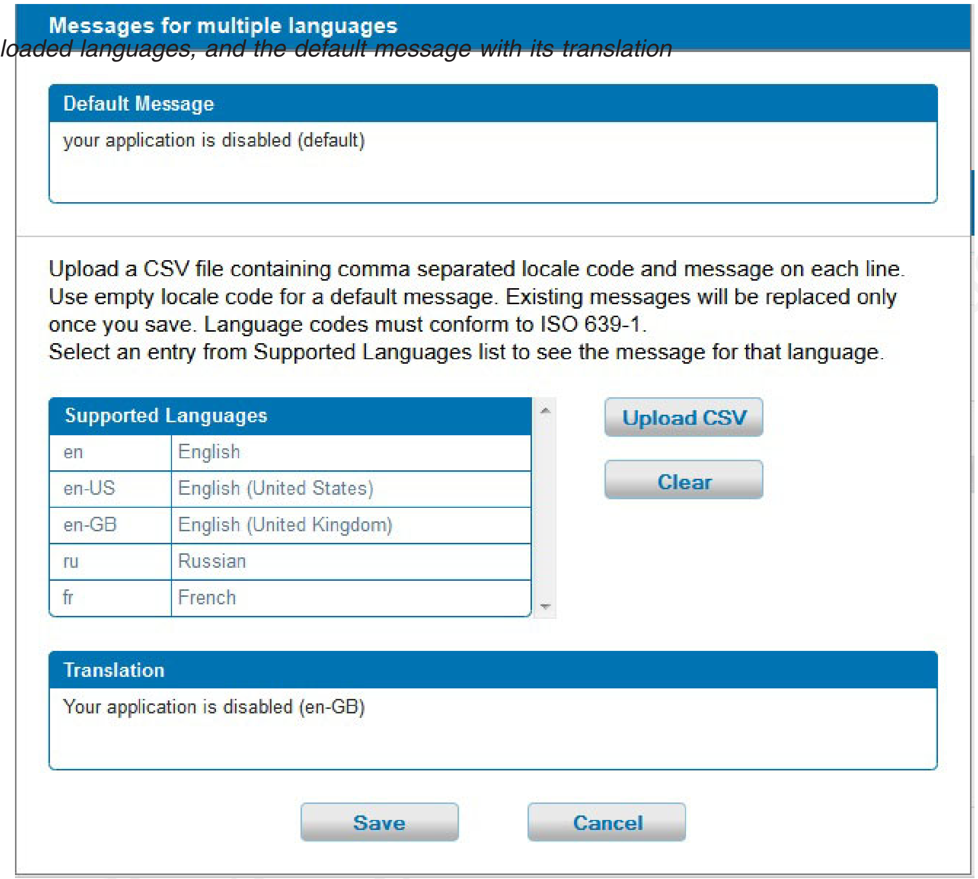


```
,your application is disabled (default)
en,Your application is disabled
en-US,Your application in disabled in US
en-GB,Your application is disabled in GB
ru,аппликация была выключена
fr,votre application est désactivée
he,האפליקציה חסומה
ja,あなたのアプリケーションが無効になっていた
```

Figure 125. 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.



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

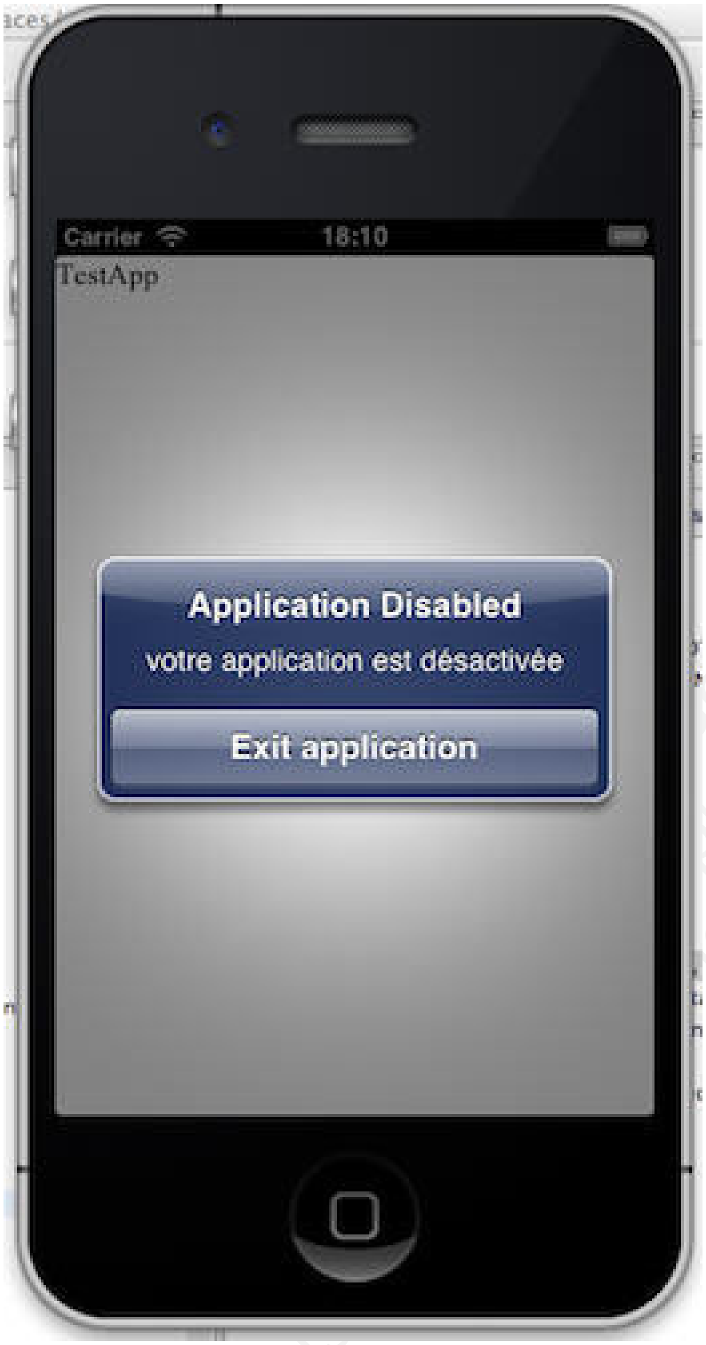
Translation

Your application is disabled (en-GB)

Figure 126. View of your uploaded languages, and the default message with its translation

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.



Carrier 18:10

TestApp

Application Disabled

votre application est désactivée

This figure displays the mobile device of the user, which shows the localized message. The title and the button caption are in English. If the locale does not supply any messages, the default message is returned.

Figure 127. Application Disabled message

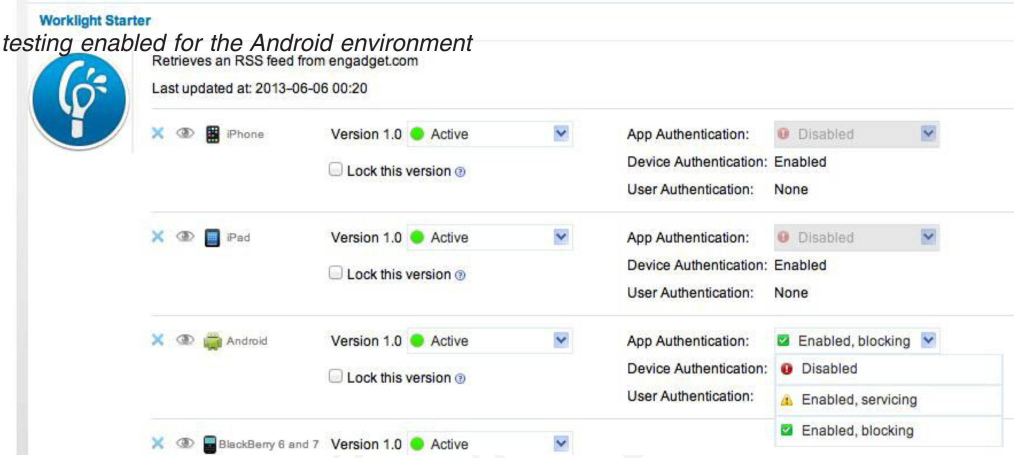
Controlling authenticity testing for an app

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 600 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.



Worklight Starter

Retrieves an RSS feed from engadget.com

Last updated at: 2013-06-06 00:20

iPhone — Version 1.0 ● Active — □ Lock this version — App Authentication: Disabled — Device Authentication: Enabled — User Authentication: None

iPad — Version 1.0 ● Active — □ Lock this version — App Authentication: Disabled — Device Authentication: Enabled — User Authentication: None

Android — Version 1.0 ● Active — □ Lock this version — App Authentication: ☑ Enabled, blocking — Device Authentication: — User Authentication:

Disabled
Enabled, servicing
Enabled, blocking

BlackBerry 6 and 7 — Version 1.0 ● Active

Figure 128. Authenticity testing enabled for the Android environment

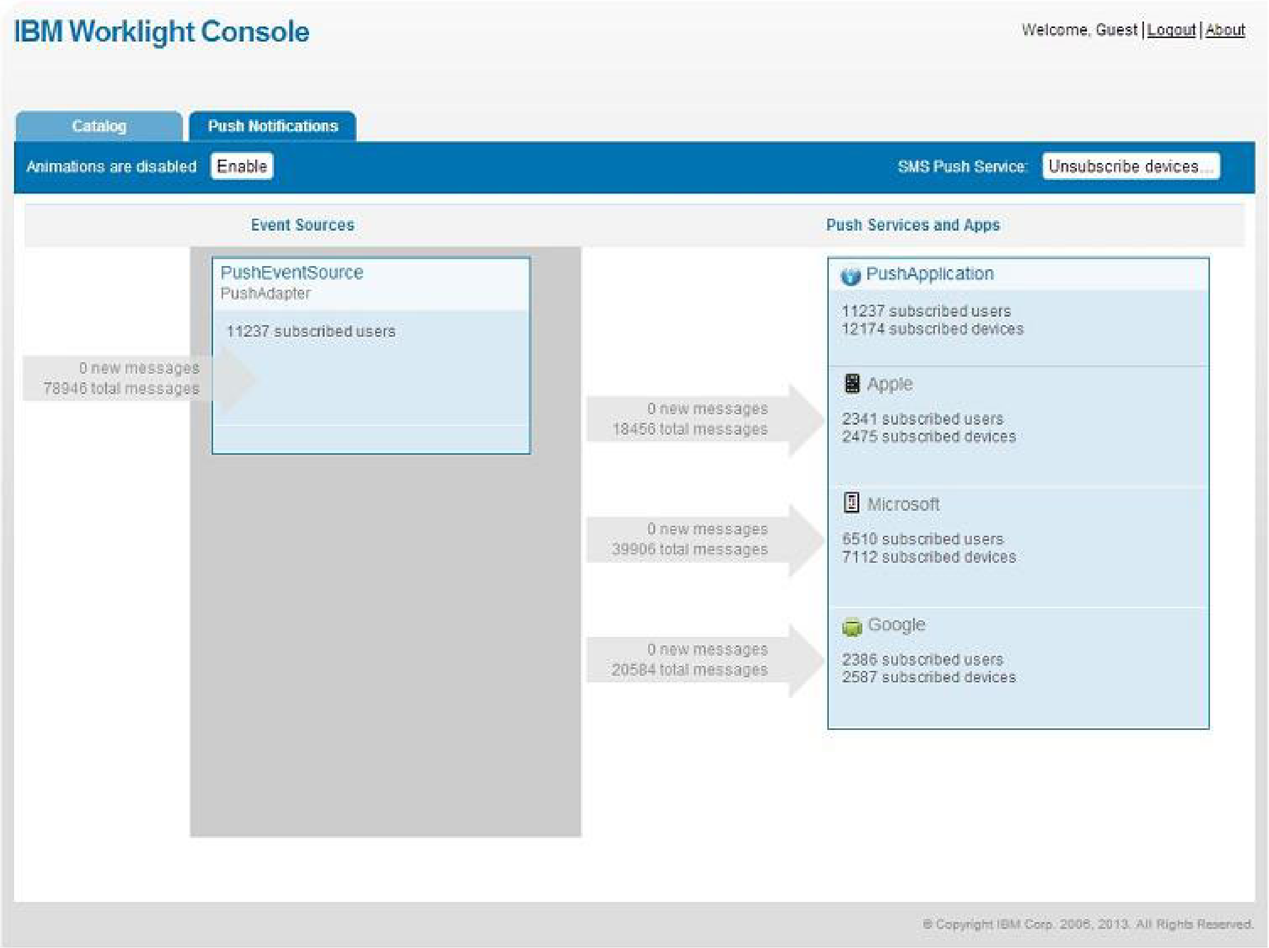
The menu has three options:

- **`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.



PushEventSource
PushAdapter
11237 subscribed users

0 new messages
78946 total messages

0 new messages
18456 total messages

0 new messages
39906 total messages

0 new messages
20584 total messages

PushApplication
11237 subscribed users
12174 subscribed devices

Apple
2341 subscribed users
2475 subscribed devices

Microsoft
6510 subscribed users
7112 subscribed devices

Google
2386 subscribed users
2587 subscribed devices

Figure 129. 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.

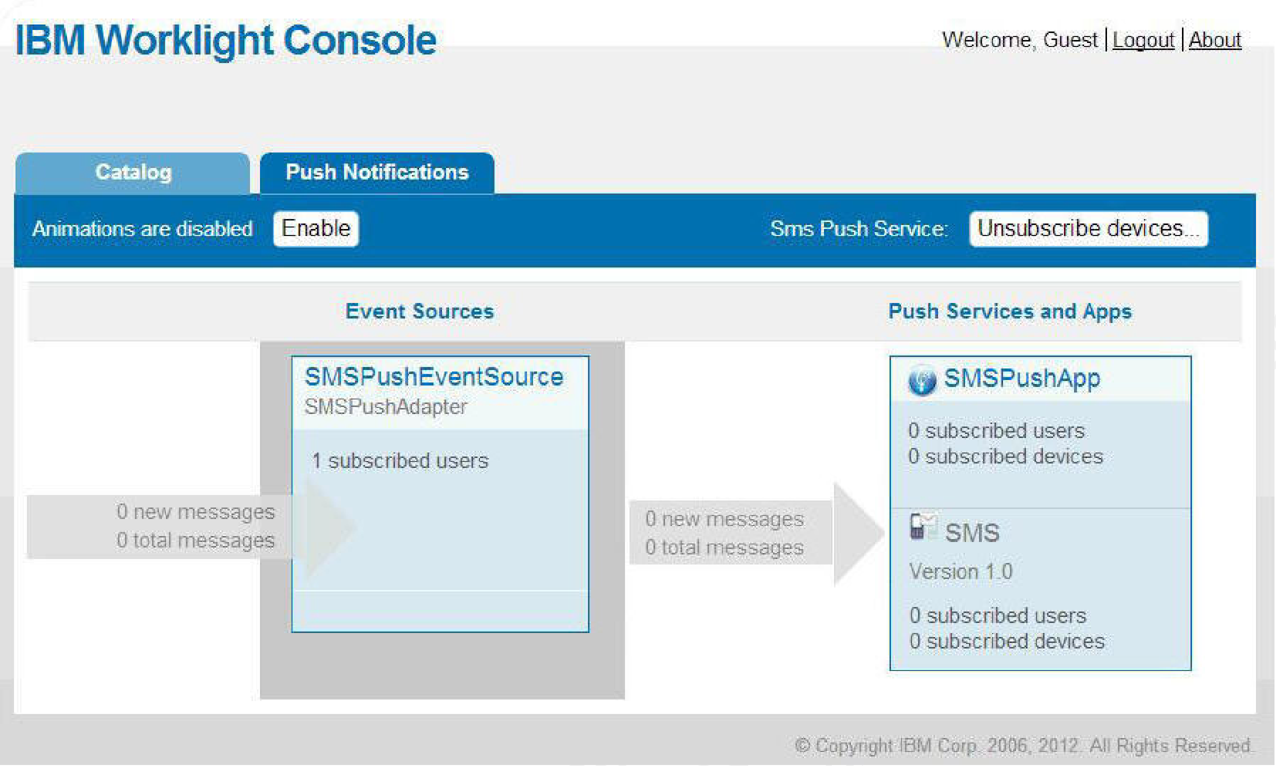
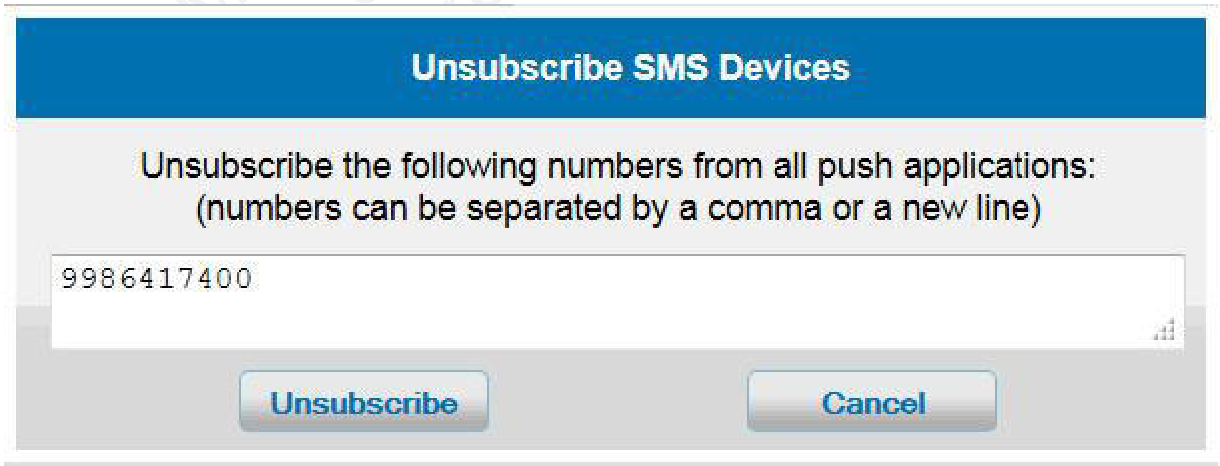


Figure 130. SMS push notifications in the Worklight Console

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.



Unsubscribe the following numbers from all push applications:
(numbers can be separated by a comma or a new line)

9986417400

Figure 131. 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, the Windows Phone 8 platform, and the BlackBerry platform for OS versions 6 and 7. Windows Phone 7, Windows RT, and BlackBerry OS 10 are not supported by the current version of the Application Center.

The Application Center manages mobile applications; it supports any kind of Android, iOS, Windows Phone 8, or BlackBerry OS 6 or OS 7 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, Windows Phone, 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.

Specific platform requirements

Different operating systems impose specific requirements for deploying, installing, or using applications on the appropriate mobile devices.

Android
: The mobile device must be configured for installation from unknown sources. The corresponding toggle can be found in the Android Settings. See User Opt-in for apps from unknown sources for details.

iOS
: All applications managed through the Application Center must be packaged for “Ad Hoc Distribution”. With an iOS developer account, you can share your application with up to 100 iOS devices. With an iOS enterprise account, you can share your in-house application with an unlimited number of iOS devices. See iOS Developer Program and iOS Enterprise Program for details.

BlackBerry
: Applications must be signed with a signing key for “BlackBerry OS 7 and earlier” that can be obtained by BlackBerry. Unsigned applications cannot access the full BlackBerry API of the device. Therefore, only very simple applications do not require this signing process. See BlackBerry Keys Order Form for details.

Windows Phone 8
: Applications are not installed from the Windows Store, but from the Application Center, which acts as what Microsoft documentation calls a “Company Hub”. See Company app distribution for Windows Phone for details.

  To use a company hub, Windows Phone requires you to register a company account with Microsoft and to sign all applications, including the Application Center client, with the company certificate. Only signed applications can be managed through the Application Center.

  You must enroll all mobile devices through an application enrollment token associated with your company account.The Application Center helps you to enroll devices through facilities to distribute the application enrollment token. See “Application enrollment tokens in Windows Phone 8” on page 886 for details.

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 865.

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 896.

The following figure shows an overview of the architecture.

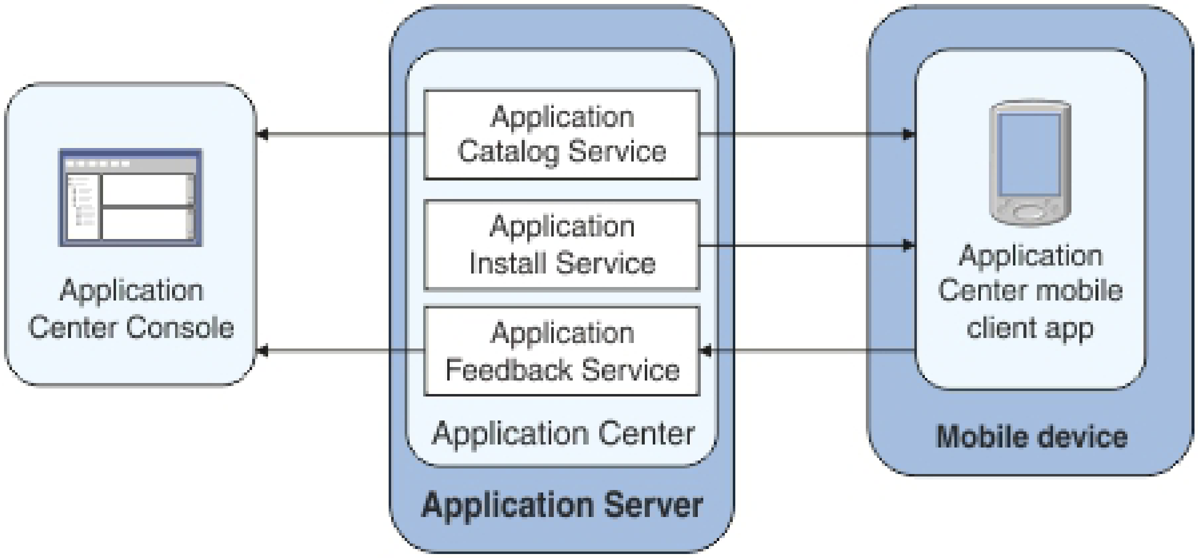

Application Center Console
Application Catalog Service
Application Install Service
Application Feedback Service
Application Center
Application Server
Application Center mobile client app
Mobile device

Figure 132. 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, Windows Phone 8, and BlackBerry devices. Therefore, the mobile client comes in several versions: an Android, an iOS, a Windows Phone 8, 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 138 or, if you are using LDAP authentication, “Managing users with LDAP” on page 144.

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 865.

To install and run the mobile client on:

- Android operating system: see “Installing the client on an Android mobile device” on page 896
- iOS operating system: see “Installing the client on an iOS mobile device” on page 899
- BlackBerry OS 6 and OS 7: see “Installing the client on a BlackBerry mobile device” on page 900.
- Windows Phone 8: see “Installing the client on Windows Phone 8” on page 901

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, Windows Phone, 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 the Android, the iOS, and the Windows Phone 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. The IBMApplicationCenter.apk file is 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 860 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 V5.0.

The Windows Phone version of the mobile client is included as an unsigned Windows Phone application package (.xap) file in the software delivery. The IBMApplicationCenterUnsigned.xap file is in the ApplicationCenter/installer directory.

The unsigned .xap file cannot be used directly. You must sign it with your company certificate obtained from Symantec/Microsoft **before** you can install it on a device.

Optional: If necessary, you can also build the Windows Phone version from sources.

To build the Windows Phone version, you must have the latest version of the Microsoft Visual Studio development tools.

The BlackBerry version is included as an archive (.zip) file. The IBMApplicationCenterBB6.zip file is in the ApplicationCenter/installer directory.

Optional: If necessary, you can also build the BlackBerry version from sources by using 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, iOS, Windows Phone)

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 **`IBMAppCenter`** project.
4. Select “IBMAppCenter project”.
5. Select “Copy projects into workspace”. This selection creates a copy of the project in your workspace. On UNIX systems, the **`IBMAppCenter`** project is read only at the original location. so copying projects into workspace avoids problems with file permissions.
6. Click **Finish** to import the **`IBMAppCenter`** project into Worklight Studio.

What to do next

Build the **`IBMAppCenter`** project. The Worklight project contains a single application named `AppCenter`. Right-click the application and select **Run as** > **Build All Environments**.

Android

Worklight Studio generates a native Android project in `IBMAppCenter/apps/AppCenter/android/native`. A native Android development tools (ADT) project is in the `android/native` folder. You can compile and sign this project by using the ADT tools. This project requires Android SDK level 16 to be installed, so that the resulting APK is compatible with all Android versions 2.3 and later. If you choose a higher level of the Android SDK when you build the project, the resulting APK will not be compatible with Android version 2.3.

See the Android site for developers 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 861 for more information.

iOS

Worklight Studio generates a native iOS project in `IBMAppCenter/apps/AppCenter/iphone/native`. The `IBMAppCenterAppCenterIphone.xcodeproj` file is in the `iphone/native` folder. This file is the Xcode project that you must compile and sign by using Xcode.

See The Apple developer site to learn more about how to sign the iOS mobile client application. To sign an iOS application, you must change the Bundle Identifier of the application to a bundle identifier that can be used with the provisioning profile that you use. The value is defined in the Xcode project settings as `com.`*`your_internet_domain_name`*`.appcenter`, where *your_internet_domain_name* is the name of your internet domain.

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 861 for more information.

Windows Phone 8

Worklight Studio generates a native Windows Phone 8 project in `IBMAppCenter/apps/AppCenter/windowsphone8/native`. The `AppCenter.csproj` file is in the `windowsphone8/native` folder. This file is the Visual Studio project that you must compile and sign by using Visual Studio.

See Windows Phone Dev Center to learn more about how to build and sign the Windows Phone mobile client application.

See Developing IBM Worklight applications for more information about how you can create hybrid mobile applications with Worklight Studio.

Android, iOS, Windows Phone: 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 `IBMAppCenter/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 177. Properties in the config.js file

Property	Description
`url`	The hardcoded address of the Application Center server. If this property is set, the address fields of the Login view are not displayed.
`defaultPort`	If the **`url`** property is null, this property prefills the **`port`** field of the Login view on a phone. This is a default value; the field can be edited by the user.
`defaultContext`	If the **`url`** property is null, this property prefills the **`context`** field of the Login view on a phone. This is a default value; the field can be edited by the user.
`ssl`	The default value of the SSL switch of the Login view.
`allowDowngrade`	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,

Table 177. Properties in the config.js file (continued)

Property	Description
showPreviousVersions	This property indicates whether the device user can show the details of all the versions of applications or only details of the latest version.
showInternalVersion	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.
listItemRenderer	This property can have one of these values: • full, the default value; the application lists show application name, rating, and latest version. • simple: the application lists show the application name only.
listAverageRating	This property can have one of these values: • latestVersion: the application lists show the average rating of the latest version of the application. • allVersions: the application lists show the average rating of all versions of the application.
requestTimeout	This property indicates the timeout in milliseconds for requests to the Application Center server.
gcmProjectId	The Google API project ID (project name = `com.ibm.appcenter`), which is required for Android push notifications; for example, 123456789012.
allowAppLinkReview	This property indicates whether local reviews of applications from external application stores can be registered and browsed in the Application Center. These local reviews are not visible in the external application store. These reviews are stored in the Application Center server.

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/drawable`*`density`* directories; one directory exists for each density.

iOS: Files named `icon`*`size`*`.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directory.

Windows Phone: Files named `ApplicationIcon.png`, `IconicTileSmallIcon.png`, and `IconicTileMediumIcon.png` in the `IBMAppCenter/apps/AppCenter/windowsphone8/native` directory.

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.

Windows Phone: Edit the **`Title`** attribute of the **`App`** entry in the `IBMAppCenter/apps/AppCenter/windowsphone8/native/Properties/WMAppManifest.xml` 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` directory.

Windows Phone: Edit the file named SplashScreenImage.png in the `IBMAppCenter/apps/AppCenter/windowsphone8/native` directory.

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 `IBMAppCenterBlackBerry6/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 178. Properties in the appcenter.properties file

Property	Description
`defaultServer`	The default value of the **`server`** field of the Login view. The field can be edited by the user.
`defaultPort`	The default value of the **`port`** field of the Login view. The field can be edited by the user.
`defaultContext`	The default value of the **`context`** field of the Login view. The field can be edited by the user.
`defaultUseSSL`	The default value of the SSL switch of the Login view.
`serverSettingVisibleInLoginScreen`	This property indicates whether the server, port, and context fields and the SSL check box are visible in the login screen. If this property is disabled, the **`defaultServer`**, **`defaultPort`**, **`defaultContext`**, and **`defaultUseSSL`** properties must be set, because the user cannot edit their values when they are not visible.

Table 178. Properties in the appcenter.properties file (continued)

Property	Description
`KeepLoginCredentialsTime`	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.
`listAverageRating`	This property can have one of these values: • latestVersion: the application lists show the average rating of the latest version of the application. • allVersions: the application lists show the average rating of all versions of the application.
`AdaptAppCatalogInfoLineToSorting`	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, Windows Phone, 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, Windows Phone application (.xap) files, and BlackBerry Web directory archive (.zip) files to the Application Center.

Follow the steps described in “Adding a mobile application” on page 868 to add the mobile client application for Android, iOS, Windows Phone, 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 the related task that corresponds to the version of the mobile client app determined by the operating system.

Related tasks:

“Installing the client on an Android mobile device” on page 896
You can install the mobile client, or any signed application marked with the installer flag, on your Android mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

“Installing the client on an iOS mobile device” on page 899
You can install the mobile client, or any signed application marked with the installer flag, on your iOS mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

“Installing the client on Windows Phone 8” on page 901
You can install the mobile client, or any signed application marked with the installer flag, on Windows Phone 8 by entering the access URL in your browser, entering your credentials, and completing the required steps. The company account must be preinstalled on your mobile device.

“Installing the client on a BlackBerry mobile device” on page 900
You can install the mobile client, or any signed application marked with the installer flag, on your BlackBerry mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

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.

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 variables *ibm.appcenter.push.schedule.period.amount* and *ibm.appcenter.push.schedule.unit*.

The value of *ibm.appcenter.push.schedule.period.amount* is an integer. The value of *ibm.appcenter.push.schedule.unit* can be “seconds”, “minutes”, or “hours”. If the unit is not specified, the amount is the interval expressed in hours. These variables are used to define the elapsed time between two batches of push messages.

Use JNDI properties to define these variables.

Important: In production, you should avoid setting the unit to “seconds”. The shorter the elapsed time, the higher the load on the server; the unit expressed in seconds is only implemented for testing and evaluation purposes. For example, when the elapsed time is set to 10 seconds, push messages are sent almost immediately.

See “List of JNDI properties for the Application Center” on page 172 for a complete list of properties that you can set.

Example for Apache Tomcat server

Define these variables with JNDI properties in the `server.xml` file:

```
<Environment name="ibm.appcenter.push.schedule.period.unit" override="false" type="java.lang.Strin
<Environment name="ibm.appcenter.push.schedule.period.amount" override="false" type="java.lang.Str
```

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.

Important: The Application Center client without Google Cloud Messaging: The Application Center relies on the availability of the Google Cloud Messaging (GCM) API. This API might not be available on devices in some territories such as China. To support those territories, you can build a version of the Application Center client that does not depend on the GCM API. The push notification feature does not work on that version of the Application Center client. See "Building a version of the mobile client that does not depend on the GCM API" on page 864 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.ecli
<Environment name="ibm.appcenter.gcm.signature.googleapikey" override="false" type="java.lang.Str
value="AIxaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs"/>
</Context>
```

   The JNDI property must be defined in accordance with your application server requirements.

See “List of JNDI properties for the Application Center” on page 172 for a complete list of properties that you can set.

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 179. JNDI properties

JNDI Property	Type and description	Example for Apache Tomcat server

`ibm.appcenter.apns.p12.certificate.location`	A string value that defines the full path to the `.p12` certificate.	`<Environment name="`*`ibm.appcenter.apns.p12.certificate.loca`*
		`override="false" type="java.lang.String" value=`
		`"/Users/someUser/someDirectory/apache-tomcat/conf/AppCente`

`ibm.appcenter.apns.p12.certificate.password`	A string value that defines the password needed to access the certificate.	`<Environment name="`*`ibm.appcenter.apns.p12.certificate.pass`*
		`type="java.lang.String"`
		`value="`*`this_is_a_secure_password`*`"/>`

`ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate`	A Boolean value (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="ibm.appcenter.apns.p12.certificate.isDe`
		`override="false" type="java.lang.String"`
		`value="true"/>`

See "List of JNDI properties for the Application Center" on page 172 for a complete list of properties that you can set.

Building a version of the mobile client that does not depend on the GCM API

You can remove the dependency on Google Cloud Messaging (GCM) API from the Android version of the client to comply with constraints in some territories. Push notifications do not work on this version of the client.

About this task

The Application Center relies on the availability of the Google Cloud Messaging (GCM) API. This API might not be available on devices in some territories such as China. To support those territories, you can build a version of the Application Center client that does not depend on the GCM API. The push notification feature does not work on that version of the Application Center client.

Procedure

1. Check that push notifications are disabled by checking that the `IBMAppCenter/apps/AppCenter/common/js/appcenter/config.json` file contains this line: `"gcmProjectId": "" ,`.
2. Remove from two places in the `IBMAppCenter/apps/AppCenter/android/native/AndroidManifest.xml` file all the lines that are located between these comments: `<!-- AppCenter Push configuration -->` and `<!-- end of AppCenter Push configuration -->`.
3. Delete the IBMAppCenter/apps/AppCenter/android/native/src/com/ibm/appcenter/GCMIntenteService.java class.
4. In Eclipse, run "Build Android Environment" in the `IBMAppCenter/apps/AppCenter/android` folder.
5. Delete the `IBMAppCenter/apps/AppCenter/android/native/libs/gcm.jar` file that was created by the IBM Worklight plug-in when you ran the previous "Build Android Environment" command.
6. Refresh the newly created `IBMAppCenterAppCenterAndroid` project, so that the removal of the GCM library is taken into account.
7. Build the .apk file of the Application Center.

What to do next

The `gcm.jar` library is automatically added by the IBM Worklight Eclipse plug-in each time that the Android environment is built. Therefore, this java archive file must be deleted from the `IBMAppCenter/apps/AppCenter/android/native/libs/` directory each time that the IBM Worklight Android build process is run. Otherwise, the `gcm.jar` library is present in the resulting `appcenter.apk` file.

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 applications written for these operating systems: Android, iOS, BlackBerry OS 6 and OS 7, or Windows Phone 8.
- 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.

Multicultural support: the user interface of the Application Center console has not been translated.

Starting the Application Center console

You can start the Application Center with your web browser and log in if you have the administrator role.

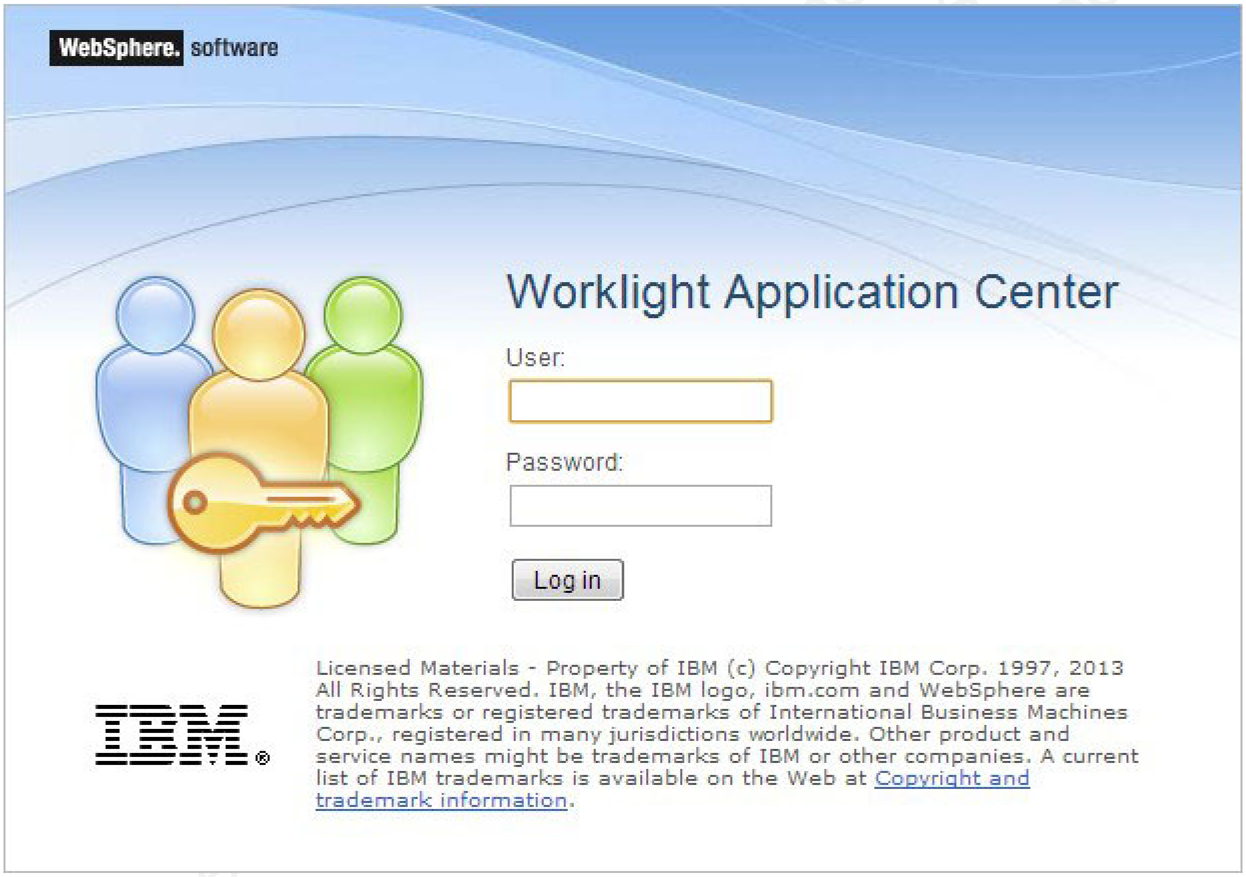
Procedure

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.



Licensed Materials - Property of IBM (c) Copyright IBM Corp. 1997, 2013 All Rights Reserved. IBM, the IBM logo, ibm.com and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information.

Figure 133. Login of the Application Center console

Note:

Only users with the administrator role can log in to the Application Center console.

Troubleshooting a corrupt login page (Apache Tomcat)

You can recover from a corrupt login page of the Application Center console when the Application Center is running in Apache Tomcat.

Symptom

When the Application Center is running in Apache Tomcat, the use of a wrong user name or password might corrupt the login page of the Application Center console.

When you try to log in to the console with an incorrect user name or an incorrect password, you receive an error message. When you correct the user name or password, instead of a successful login, you have one of the following errors; the message depends on your web browser.

- The same error message as before
- The message “The connection was reset”
- The message “The time allowed for login exceeded”

Cause

The behavior is linked to the management by Apache Tomcat of the `j_security_check` servlet. This behavior is specific to Apache Tomcat and does not occur in any of the WebSphere Application Server profiles.

Solution

The workaround is to click the refresh button of the browser to refresh the web page after a login failure. Then, enter the correct credentials.

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.

Click **Applications** to access Application Management.

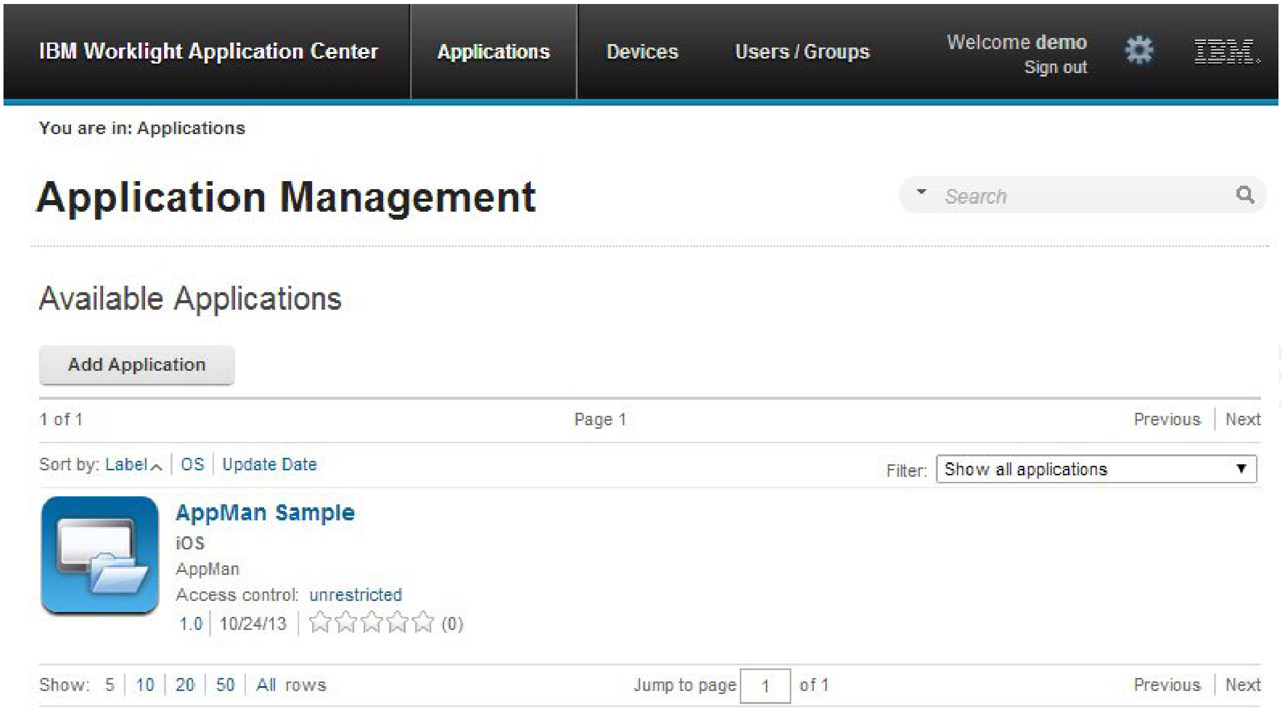
Application Center installed on WebSphere Application Server Liberty Profile or on Apache Tomcat

Installations of the Application Center on these application servers, during installation of Worklight with the IBM 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.



AppMan Sample
iOS
AppMan
Access control: unrestricted
1.0 | 10/24/13 | ☆☆☆☆☆ (0)

Figure 134. Available applications

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 OS 6 and OS 7

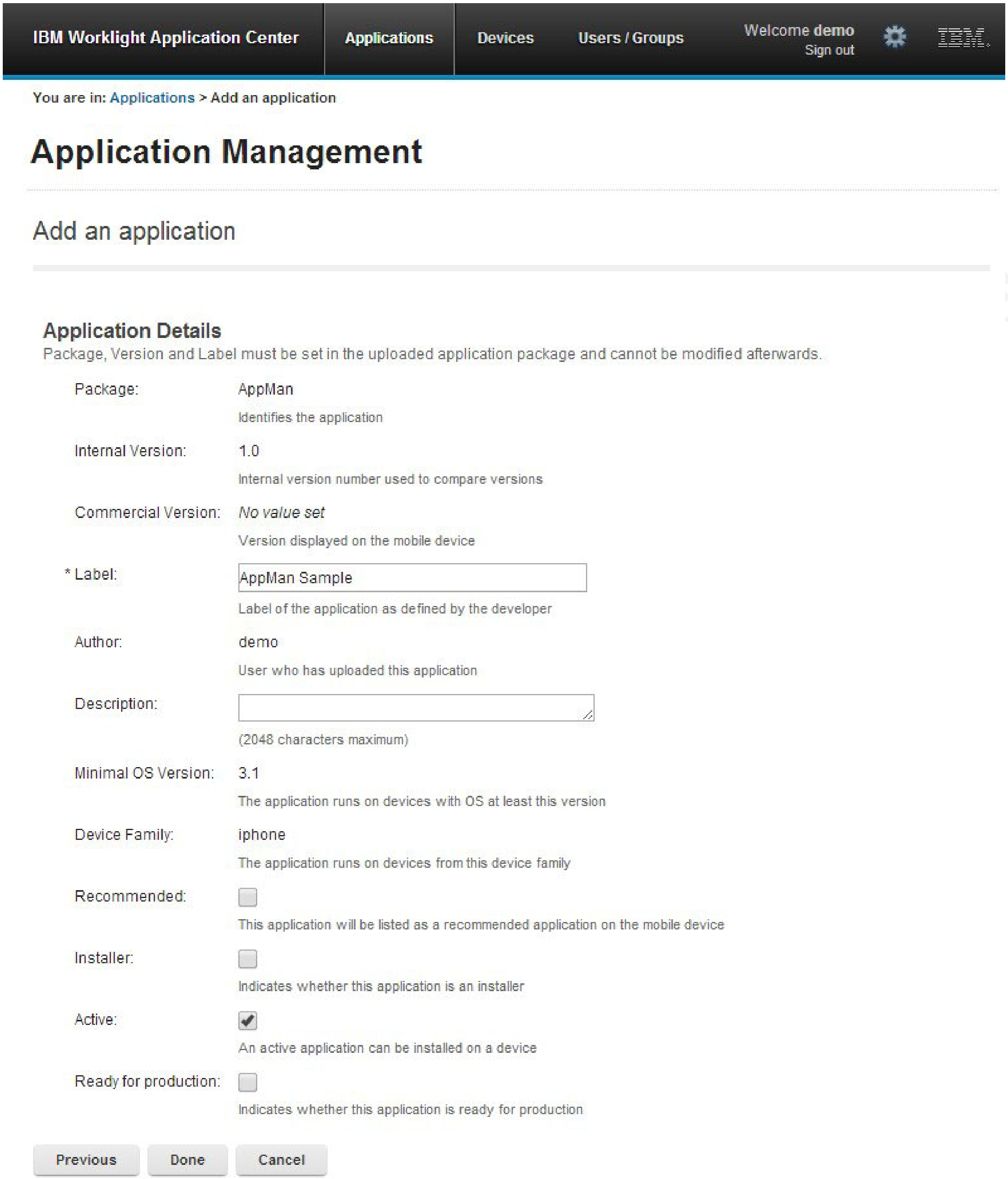
The application file extension is `zip`. This archive file must contain a file with extension `jad` and all related files with extension `cod`. If you are using the BlackBerry Eclipse IDE for a native application, the files are in the `deliverables/Web` folder. You can place the entire folder in an archive (.zip) file.

If you are using the Ripple Environment in combination with Worklight Studio for a hybrid application, the files are in the `OTAInstall` folder. You can place the entire folder in an archive (.zip) file.

Windows Phone 8

The application file extension is `xap`. The application must be signed with a company account. The application enrollment token for this company account must be made available to Windows Phone 8 devices before the application can be installed on the devices. See “Application enrollment tokens in Windows Phone 8” on page 886 for details.

4. Click **Next** to access the properties to complete the definition of the application.
5. Complete the properties to define the application. See Application properties for information about how to complete property values.
6. Click **Finish**.



You are in: Applications > Add an application

Application Management

Add an application

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	AppMan Identifies the application
Internal Version:	1.0 Internal version number used to compare versions
Commercial Version:	*No value set* Version displayed on the mobile device
* Label:	AppMan Sample Label of the application as defined by the developer
Author:	demo User who has uploaded this application
Description:	(2048 characters maximum)
Minimal OS Version:	3.1 The application runs on devices with OS at least this version
Device Family:	iphone The application runs on devices from this device family
Recommended:	□ This application will be listed as a recommended application on the mobile device
Installer:	□ Indicates whether this application is an installer
Active:	☑ An active application can be installed on a device
Ready for production:	□ Indicates whether this application is ready for production

Figure 135. Application properties, adding an application

Adding an application from a public app store

Application Center supports adding to the catalog applications that are stored in third-party application stores, such as Google play or Apple iTunes.

About this task

Applications from third-party app stores appear in the Application Center catalog like any other application, but users are directed to the corresponding public app store to install the application. You add an application from a public app store in the console, in the same place as you add an application created within your own enterprise. See “Adding a mobile application” on page 868.

Note: Currently, the Application Center supports only Google play and Apple iTunes. Microsoft Market Place for Windows Phone 8 and BlackBerry App World are not yet supported.

Instead of providing the application executable, you must provide a URL to the third party application store where the application is stored. To make it easy to find the correct application link, the console provides direct links in the “Add an application” page to the supported third-party application store web sites.

The Google play store address is https://play.google.com/store/apps.

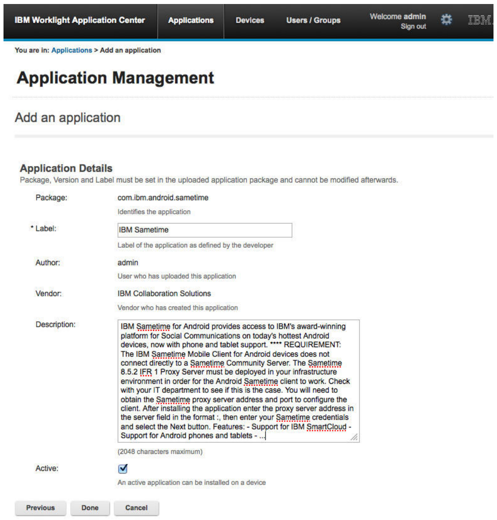
The Apple iTunes store address is https://linkmaker.itunes.apple.com/; use the linkmaker site rather than the iTunes site, because you can search this site for all kinds of iTunes items, including songs, podcasts, and other items supported by Apple. Only selecting iOS applications provides you with compatible links to create application links.

Procedure

1. Click the URL of the public app store that you want to browse.
2. Copy the URL of the application in the third-party app store to the Application URL text field in the “Add an application” page of the Application Center console.
   - **Google play**:
     a. Select an application in the store.
     b. Click the detail page of the application.
     c. Copy the address bar URL.
   - **Apple iTunes**:
     a. When the list of items is returned in the search result, select the item that you want.
     b. On the right, click “iPhone App Link” to open the application details page.
     c. Copy the address bar URL.
3. When the application link is in the Application URL text field of the console, click **Next** to validate the creation of the application link. If the validation is successful, this action will display the application properties.

   If the validation is unsuccessful, an error message will be displayed in the “Add an application” page. You can either try another link or cancel the attempt to create the current link.

   If the validation of the application link is successful, you can modify the application description in the application properties before performing the next step.



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: com.ibm.android.sametime

Identifies the application

* Label: IBM Sametime

Label of the application as defined by the developer

Author: admin

User who has uploaded this application

Vendor: IBM Collaboration Solutions

Vendor who has created this application

Description: IBM Sametime for Android provides access to IBM's award-winning platform for Social Communications on today's hottest Android devices, now with phone and tablet support. **** REQUIREMENT: The IBM Sametime Mobile Client for Android devices does not connect directly to a Sametime Community Server. The Sametime 8.5.2 IFR 1 Proxy Server must be deployed in your infrastructure environment in order for the Android Sametime client to work. Check with your IT department to see if this is the case. You will need to obtain the Sametime proxy server address and port to configure the client. After installing the application enter the proxy server address in the server field in the format :, then enter your Sametime credentials and select the Next button. Features: - Support for IBM SmartCloud - Support for Android phones and tablets - ...

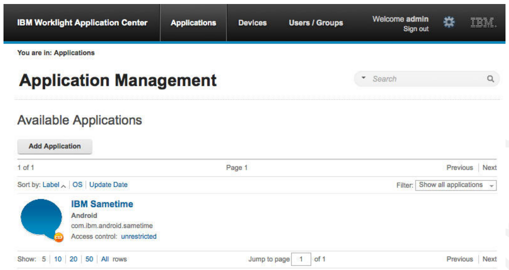
(2048 characters maximum)

Active: ☑

An active application can be installed on a device

Figure 136. Modified application description in application properties

4. Click **Done** to create the application link. This action makes the application available to the corresponding version of the Application Center mobile client. A small link icon appears on the application icon to show that this application is stored in a public app store and is different from a binary app.



You are in: Applications

Application Management

Available Applications

1 of 1 Page 1 Previous | Next

Sort by: Label | OS | Update Date Filter: Show all applications

IBM Sametime
Android
com.ibm.android.sametime
Access control: unrestricted

Show: 5 | 10 | 20 | 50 | All rows Jump to page 1 of 1 Previous | Next

Figure 137. Link to an application stored in Google play

Related concepts:

Configuring WebSphere Application Server to support applications in public app stores
Configure WebSphere Application Server full profile and Liberty profile before access to public app stores through application links, because of the use of SSL connections.

Related tasks:

Configuring WebSphere Application Server to support applications in Google play
Configure WebSphere Application Server to enable links in the Application Center console to access applications in Google play.

Configuring WebSphere Application Server to support applications in Apple iTunes
Configure WebSphere Application Server to enable links in the Application Center console to access applications in Apple iTunes.

"Installing applications through public app stores" on page 915
You can link from the mobile client to applications that are stored in supported public app stores and install these applications on your compatible device by following the normal procedure of the public app store.

Application properties

Applications have their own sets of properties that depend on the operating system on the mobile device and that cannot be edited. Applications also have a common property 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.

Properties of Windows Phone 8 applications

- **Package** is the product identifier of the application; **ProductID** attribute of the **App** element in the manifest file of the application. See Windows Phone documentation.
- **Internal Version** is the version identification of the application; **Version** attribute of the **App** element in the manifest file of the application. See Windows Phone documentation.
- **Commercial Version**, like **Internal Version**, is the version of the application.
- **Label** is the title of the application; **Title** attribute of the **App** element in the manifest file of the application. See Windows Phone documentation.
- **Vendor** is the vendor who created the application; **Publisher** attribute of the **App** element in the manifest file of the application. See Windows Phone documentation.

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 896.

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

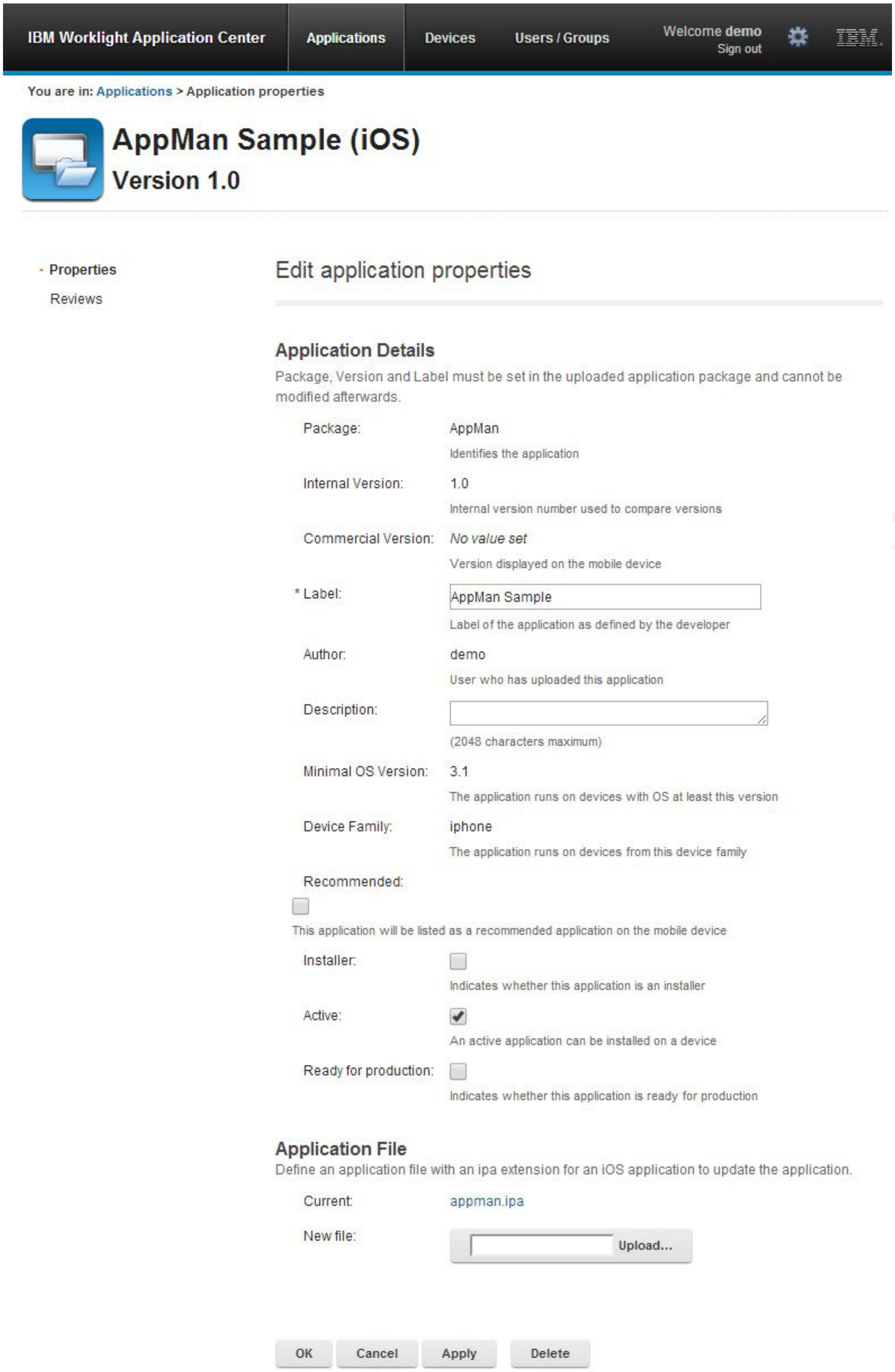
You can edit the properties of an application in the list of uploaded applications.

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 873 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.



You are in: Applications > Application properties

AppMan Sample (iOS)
Version 1.0

Edit application properties

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package: AppMan
Identifies the application

Internal Version: 1.0
Internal version number used to compare versions

Commercial Version: *No value set*
Version displayed on the mobile device

* Label: AppMan Sample
Label of the application as defined by the developer

Author: demo
User who has uploaded this application

Description:
(2048 characters maximum)

Minimal OS Version: 3.1
The application runs on devices with OS at least this version

Device Family: iphone
The application runs on devices from this device family

Recommended:
☐
This application will be listed as a recommended application on the mobile device

Installer: ☐
Indicates whether this application is an installer

Active: ☑
An active application can be installed on a device

Ready for production: ☐
Indicates whether this application is ready for production

Application File

Define an application file with an ipa extension for an iOS application to update the application.

Current: appman.ipa

New file:

Figure 138. 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.

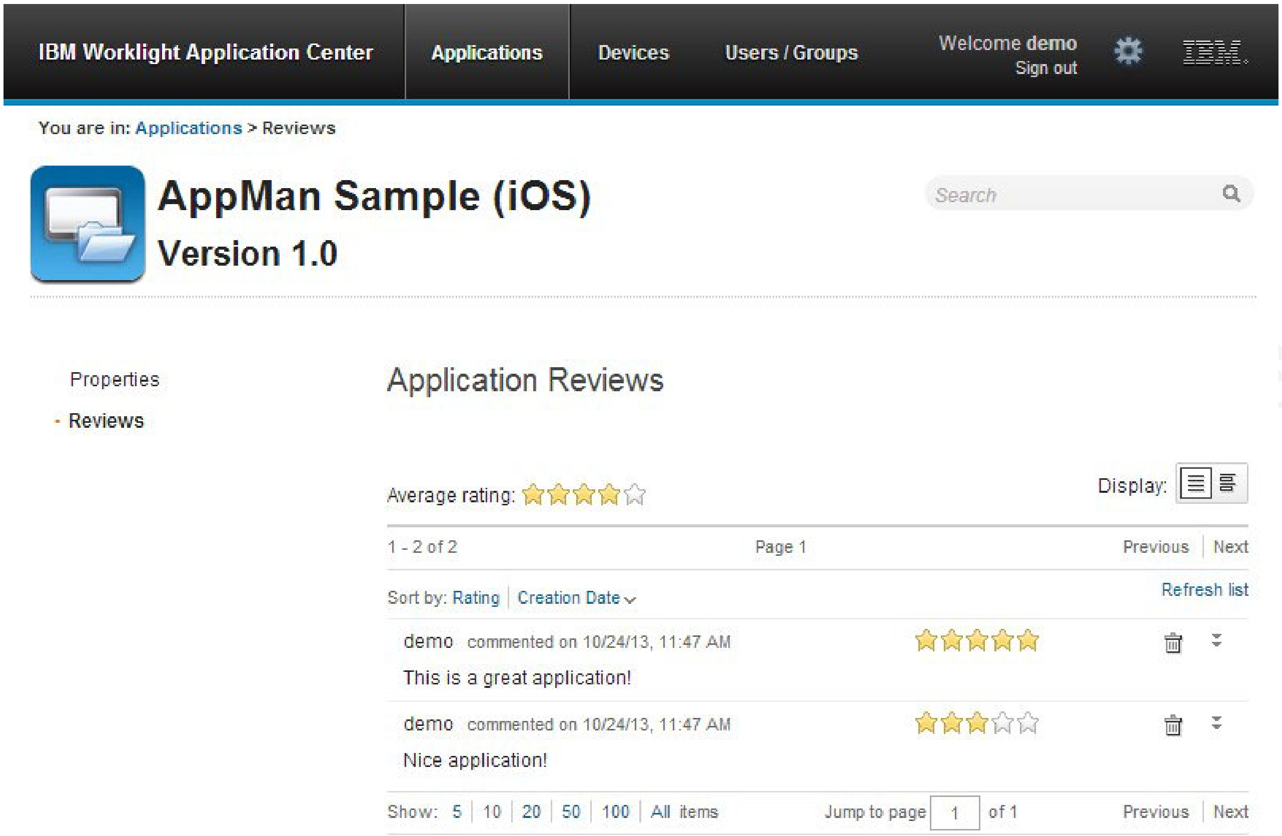
About this task

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**.



IBM Worklight Application Center | Applications | Devices | Users / Groups | Welcome demo | Sign out

You are in: Applications > Reviews

AppMan Sample (iOS)
Version 1.0

Search

Properties
- Reviews

Application Reviews

Average rating: ★★★★☆

Display:

1 - 2 of 2 | Page 1 | Previous | Next

Sort by: Rating | Creation Date | Refresh list

demo commented on 10/24/13, 11:47 AM ★★★★★
This is a great application!

demo commented on 10/24/13, 11:47 AM ★★★☆☆
Nice application!

Show: 5 | 10 | 20 | 50 | 100 | All items | Jump to page 1 of 1 | Previous | Next

Figure 139. 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.

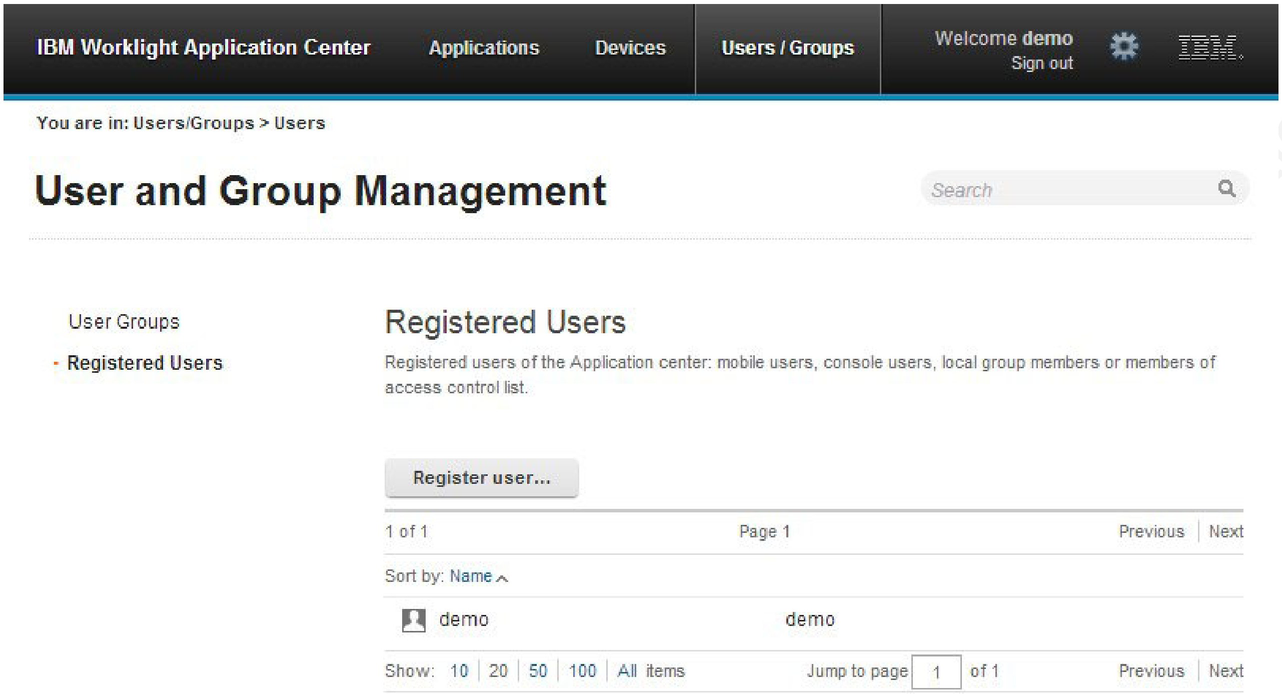
Purpose

Use users and groups in the definition of access control lists (ACL).

Managing registered users

To manage registered users, click the **Users/Groups** tab and select **Registered users**. You obtain a list of registered users of the Application Center that includes:

- Mobile client users
- Console users
- Local group members
- Members of an access control list



Registered users of the Application center: mobile users, console users, local group members or members of access control list.

Figure 140. 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.

To register new users, click **Register User**, enter the login name and the display name, and click **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:

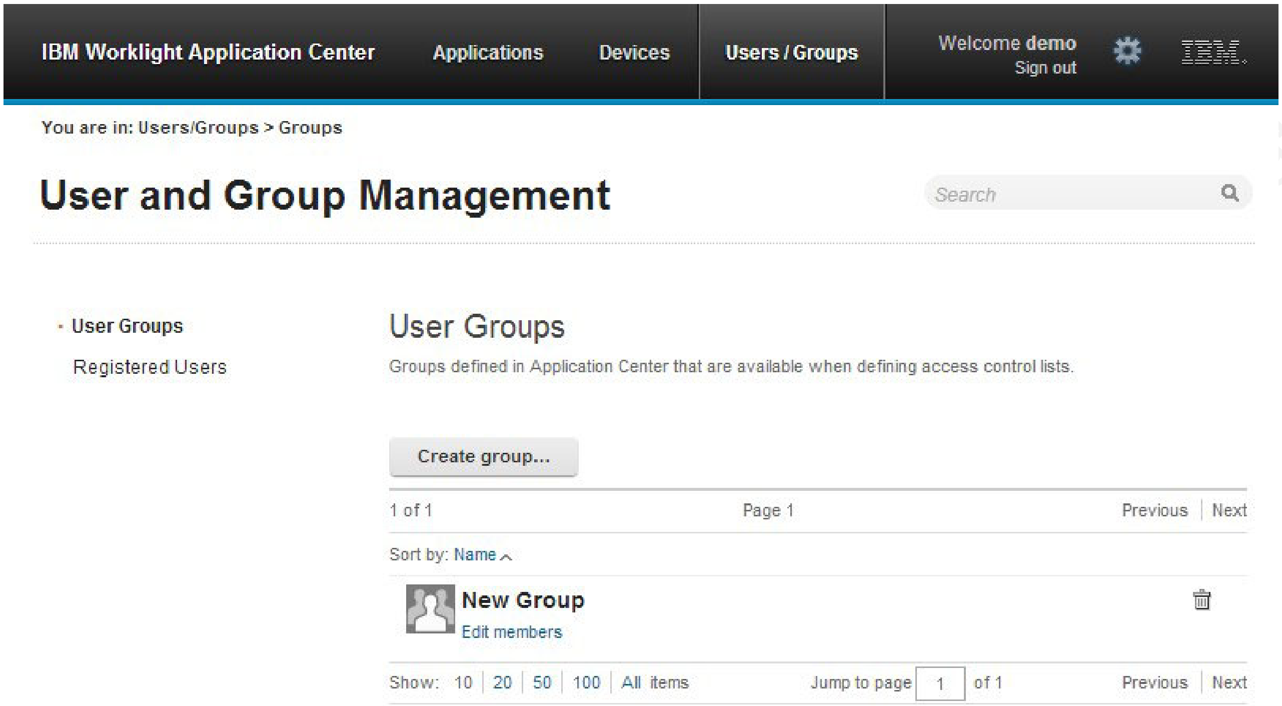
When you unregister a user, the user is not removed from the application server or the LDAP repository.

Managing local groups

To manage local groups, click the **Users/Groups** tab and select **User group**.

To create a local group, click **Create group**. Enter the name of the new group and click **OK**.

If the Application Center is connected to an LDAP repository, the search includes local groups as well as the groups defined in the LDAP repository. If the repository is not LDAP, only local groups are available to the search.

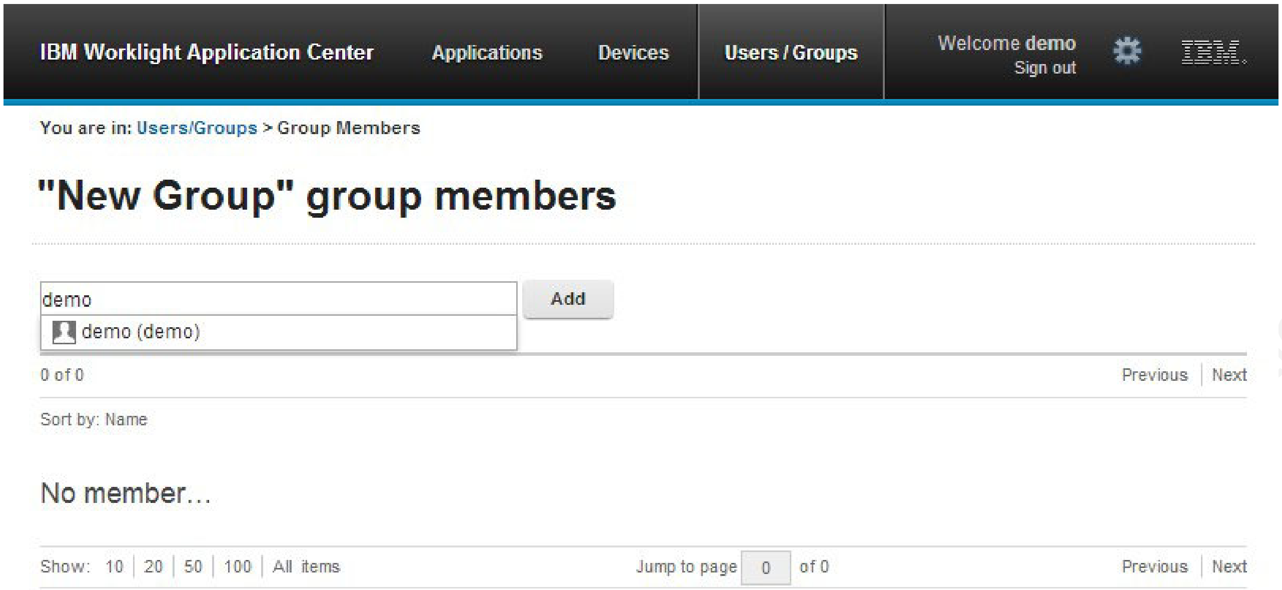


Groups defined in Application Center that are available when defining access control lists.

Figure 141. 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.


IBM Worklight Application Center
Applications
Devices
Users / Groups
Welcome demo
Sign out
You are in: Users/Groups > Group Members
"New Group" group members
Add
0 of 0
Previous
Next
Sort by: Name
No member...
Show: 10 | 20 | 50 | 100 | All items
Jump to page
of 0
Previous
Next

demo

demo (demo)

0

Figure 142. 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.

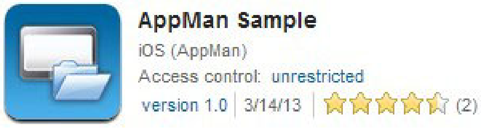
Managing access control

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.



AppMan Sample

iOS (AppMan)

Access control: unrestricted

version 1.0 | 3/14/13 | ★★★★☆ (2)

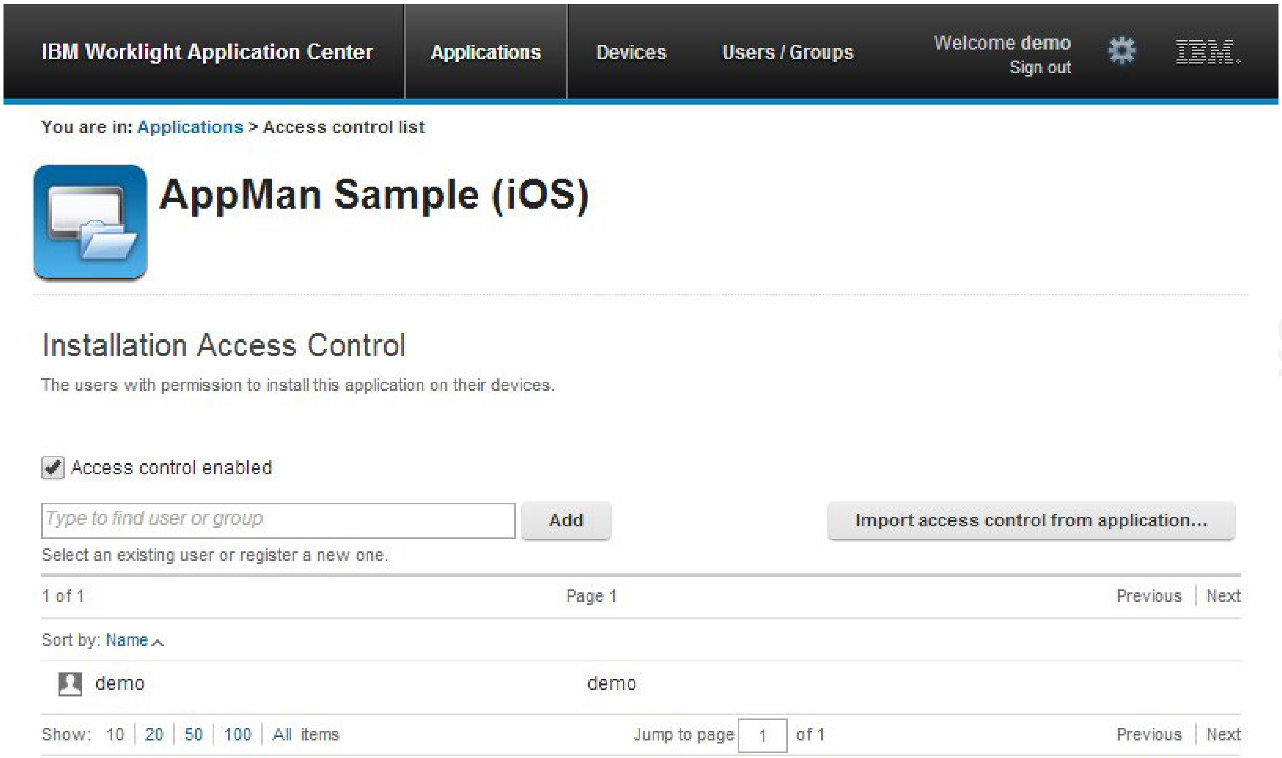
2. Select **Access control enabled** to enable access control.
3. Add users or groups to the access list.

   To add a single user or group, enter a name, select the entry in the matching entries found, and click **Add**.

   If the Application Center is connected to an LDAP repository, you can search for users and groups in the repository as well as locally defined groups. If the repository is not LDAP, you can search only local groups and registered users. Local groups are exclusively defined in the **Users/Groups** tab. When you use the Liberty profile federated registry, you can only search for users by using the login name; the result is limited to a maximum of 15 users and 15 groups (instead of 50 users and 50 groups).

   To register a user at the same time as you add the user to the access list, enter the name and click **Add**. Then you must specify the login name and the display name of the user.

   To add all the users of an application, click **Add users from application** and select the appropriate application.


IBM Worklight Application Center
Applications
Devices
Users / Groups
Welcome demo
Sign out
IBM.

You are in: Applications > Access control list

AppMan Sample (iOS)

Installation Access Control

The users with permission to install this application on their devices.

☑ Access control enabled

Type to find user or group
Add
Import access control from application...

Select an existing user or register a new one.

1 of 1 Page 1 Previous | Next

Sort by: Name

demo demo

Show: 10 | 20 | 50 | 100 | All items Jump to page 1 of 1 Previous | Next

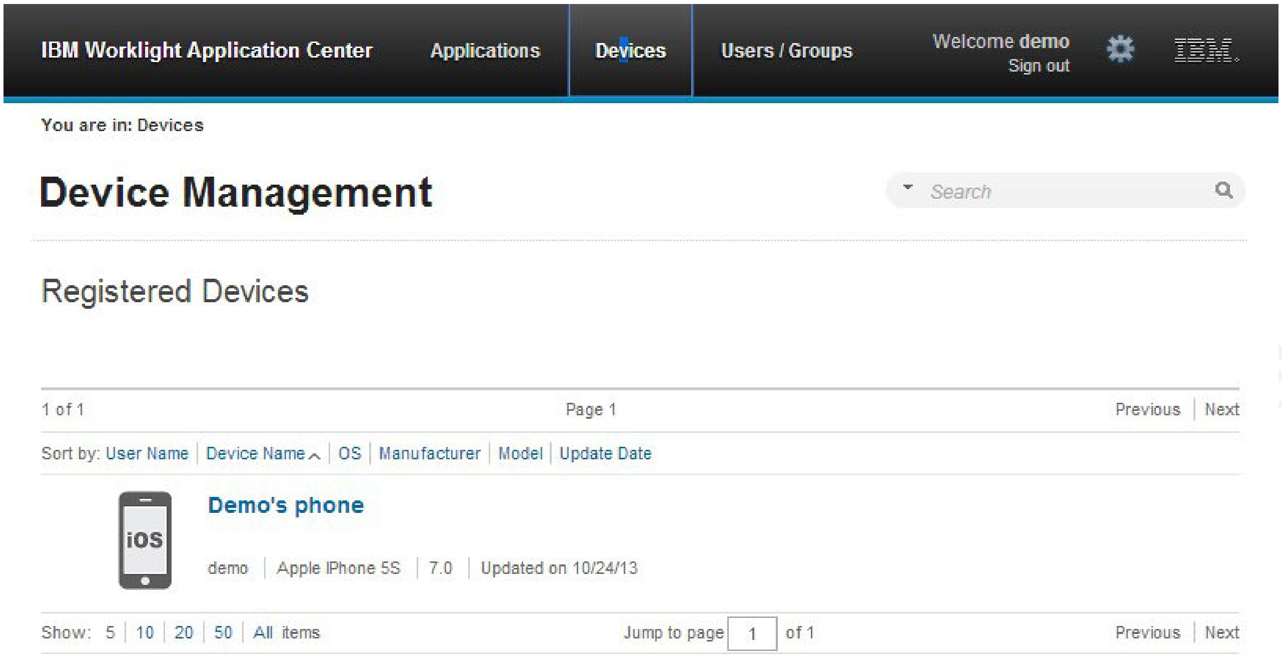
Figure 143. Adding users to the access list

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 the **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

You are in: Devices

Device Management

Search

Registered Devices

1 of 1 | Page 1 | Previous | Next

Sort by: User Name | Device Name | OS | Manufacturer | Model | Update Date

Demo's phone

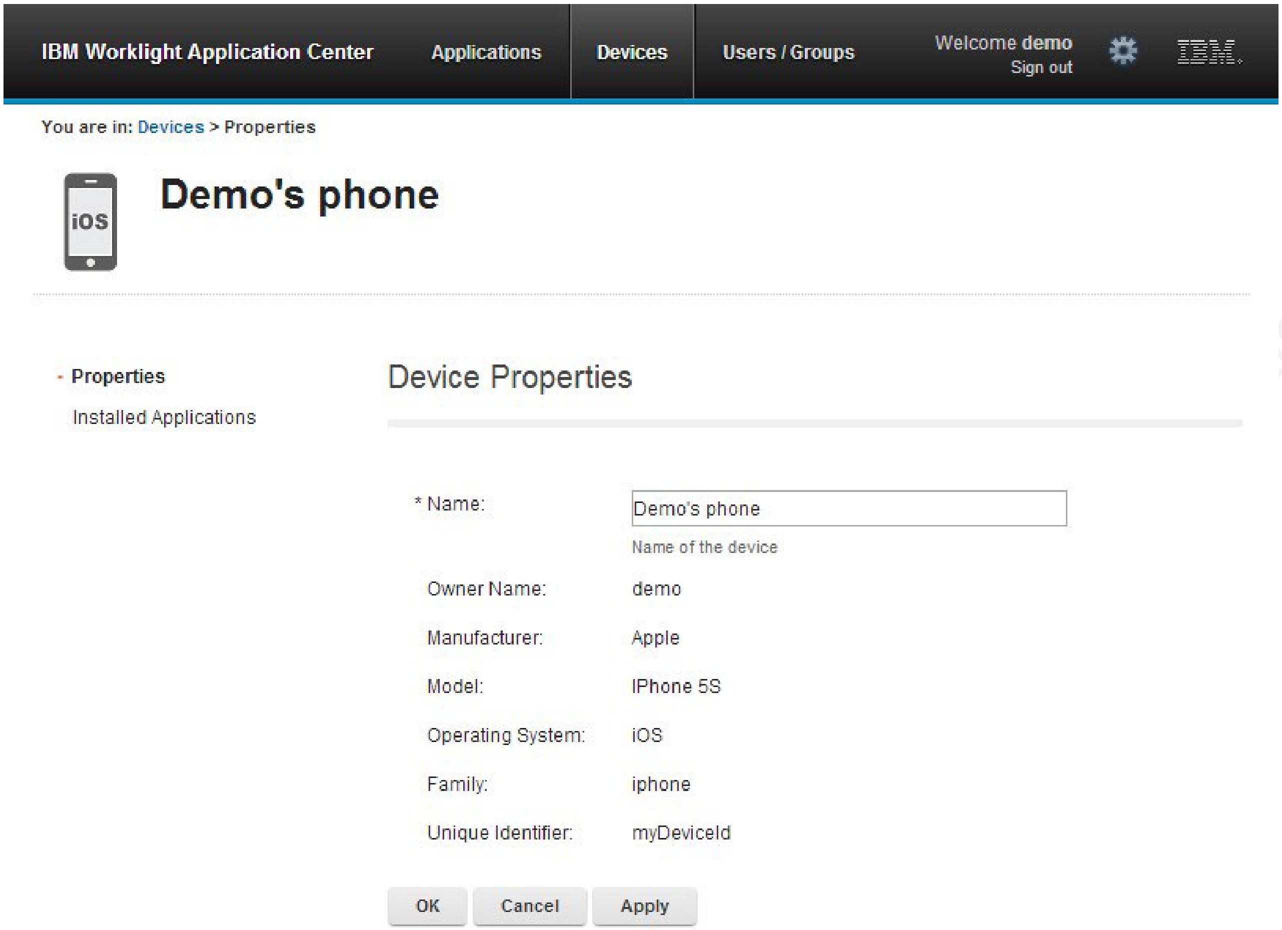
demo | Apple IPhone 5S | 7.0 | Updated on 10/24/13

Show: 5 | 10 | 20 | 50 | All items | Jump to page 1 of 1 | Previous | Next

Figure 144. The device list

Device properties

Click a device in the list of devices to view the properties of the device or the applications installed on that device.



Demo's phone

Device Properties

* Name:	Demo's phone Name of the device
Owner Name:	demo
Manufacturer:	Apple
Model:	IPhone 5S
Operating System:	iOS
Family:	iphone
Unique Identifier:	myDeviceId

Figure 145. 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.

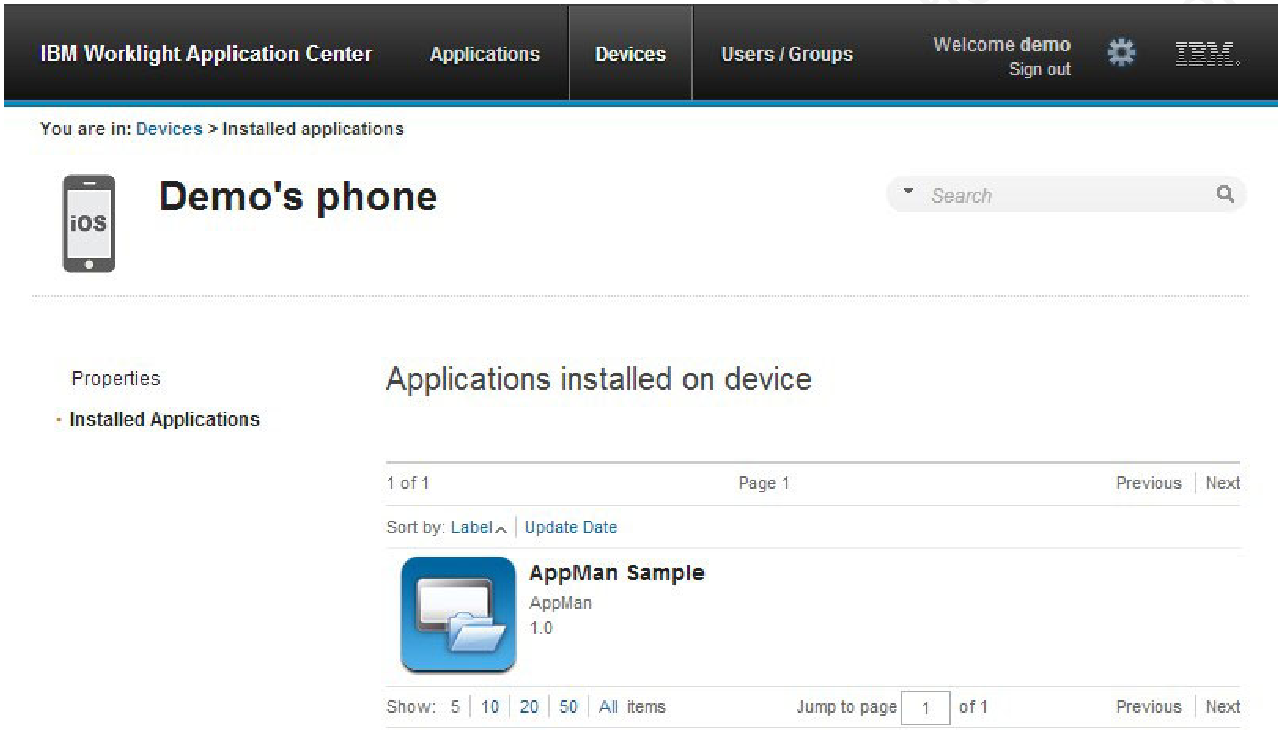


Figure 146. Applications installed on a device

Application enrollment tokens in Windows Phone 8

The Windows Phone 8 operating system requires users to enroll each device with the company before users can install company applications on their devices. One way to enroll devices is by using an application enrollment token.

Purpose

Application enrollment tokens enable you to install company applications on a Windows Phone 8 device. You must first install the enrollment token for a specified company on the device to enroll the device with the company. Then, you can install applications that are created and signed by the corresponding company.

The Application Center simplifies the delivery of the enrollment token. In your role of administrator of the Application Center catalog, you can manage the enrollment tokens from the Application Center console. Once the enrollment tokens are declared in the Application Center console, they are available for Application Center users to enroll their devices.

The enrollment tokens interface available from the Application Center console in the Settings view enables you to manage application enrollment tokens for Windows Phone 8 by registering, updating, or deleting them.

Managing application enrollment tokens

In your role of administrator of the Application Center, you can access the list of registered tokens by clicking the gear icon ⚙ in the screen header to display Application Center Settings. Then, select **Enrollment Tokens** to display the list of registered tokens.

To enroll a device, the device user must upload and install the token file **before** installing the Application Center mobile client. The mobile client is also a company application. Therefore, the device must be enrolled before the mobile client can be installed.

The registered tokens are available through the bootstrap page at `http://`*`hostname`*`:`*`portnumber`*`/appcenterconsole/installers.html`, where *hostname* is the host name of the server hosting the Application Center console and *portnumber* is the corresponding port number.

To register a token in the Application Center console, click **Upload Token** and select a token file. The token file extension is `aetx`.

To update the certificate subject of a token, select the token name in the list, change the value, and click **OK**.

To delete a token, click the trash can icon on the right side of the token in the list.

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
`-s`	**`serverpath`**	The path to the Application Center server.
`-c`	**`context`**	The context of the Application Center web application.
`-u`	**`user`**	The user credentials to access the Application Center.
`-p`	**`password`**	The password of the user.
`-d`	**`description`**	The description of the application to be uploaded.
`-l`	**`label`**	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.
`-isActive`	true or false	The application is stored in the Application Center as an active or inactive application.
`-isInstaller`	true or false	The application is stored in the Application Center with the "installer" flag set appropriately.
`-isReadyForProduction`	true or false	The application is stored in the Application Center with the "ready-for-production" flag set appropriately.

Option	Content indicated by	Description
-isRecommended	true or false	The application is stored in the Application Center with the “recommended” flag set appropriately.
-e		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 demopas
```

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	**serverpath**	The path to the Application Center server.
-c	**context**	The context of the Application Center web application.
-u	**user**	The user credentials to access the Application Center.
-p	**password**	The password of the user.

Option	Content indicated by	Description
`-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 demopa
```

Using the stand-alone tool to clear the LDAP cache

Use the stand-alone tool to clear the LDAP cache and make changes to LDAP users and groups visible immediately in the Application Center.

About this task

When the Application Center is configured with LDAP, changes to users and groups on the LDAP server become visible to the Application Center after a delay. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default, the delay is 24 hours. If you do not want to wait for this delay to expire after changes to users or groups, you can call the stand-alone tool from the command line to clear the cache of LDAP data. By using the stand-alone tool to clear the cache, the changes become visible immediately.

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 -clearLdapCache [options]
   ```

   You can pass any of the available options in the command line.

Option	Content indicated by	Description
-s	**serverpath**	The path to the Application Center server.
-c	**context**	The context of the Application Center web application.
-u	**user**	The user credentials to access the Application Center.
-p	**password**	The password of the user.
-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.

Example

In this example, user **demo** has the password **demopassword**.

```
java com.ibm.appcenter.Upload -clearLdapCache -s http://localhost:9080 -c applicationcenter -u dem
```

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.

Apache Ant is required to run these tasks. The minimum supported version of Apache Ant is listed in “System requirements for using IBM Worklight” on page 9.

For convenience, Apache Ant 1.8.4 is included in Worklight Server. In the *WL_INSTALL_DIR*/shortcuts/ directory, the following scripts are provided:

- ant for UNIX / Linux
- ant.bat for Windows

These scripts are ready to run, which means that they do not require specific environment variables. If the environment variable JAVA_HOME is set, the scripts accept it.

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
serverPath	To connect to the Application Center. The default value is http://localhost:9080.
context	The context of the Application Center. The default value is /applicationcenter.
loginUser	The user name with permissions to upload an application.
loginPass	The password of the user with permissions to upload an application.

Parameters of ant task	Description
forceOverwrite	If set to true, the ant task attempts to overwrite applications in the Application Center when it uploads an application that is already present. This parameter is only available in the upload ant task.
file	The .apk or .ipa file to be uploaded to the Application Center or to be deleted from the Application Center. This parameter has no default value.
fileset	To upload or delete multiple files.
application	The package name of the application; this parameter is only available in the delete ant task.
os	The operating system of the application (Android, iOS, BlackBerry). This parameter is only available in the delete ant task.
version	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="${upload.file}" />
    </target>
    <target name="upload.AllApps" description="Uploads all found APK and IPA files" depends="upload.init">
    <uploadapps serverPath="${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
install.dir	Defaults to `../../`
server.path	The default value is `http://localhost:9080`.
context.path	The default value is `applicationcenter`.
upload.file	This property has no default value. It must include the exact file path.
workspace.root	Defaults to `../../`
login.user	The default value is `appcenteradmin`.
login.pass	The default value is `admin`.
force	The default value is `true`..

To specify these parameters by command line when you call ant, add `-D` before the property name. For example:

```
-Dserver.path=http://localhost:8888/
```

Publishing Worklight applications to the Application Center

You can use the application management plug-in to publish native applications to the IBM Application Center.

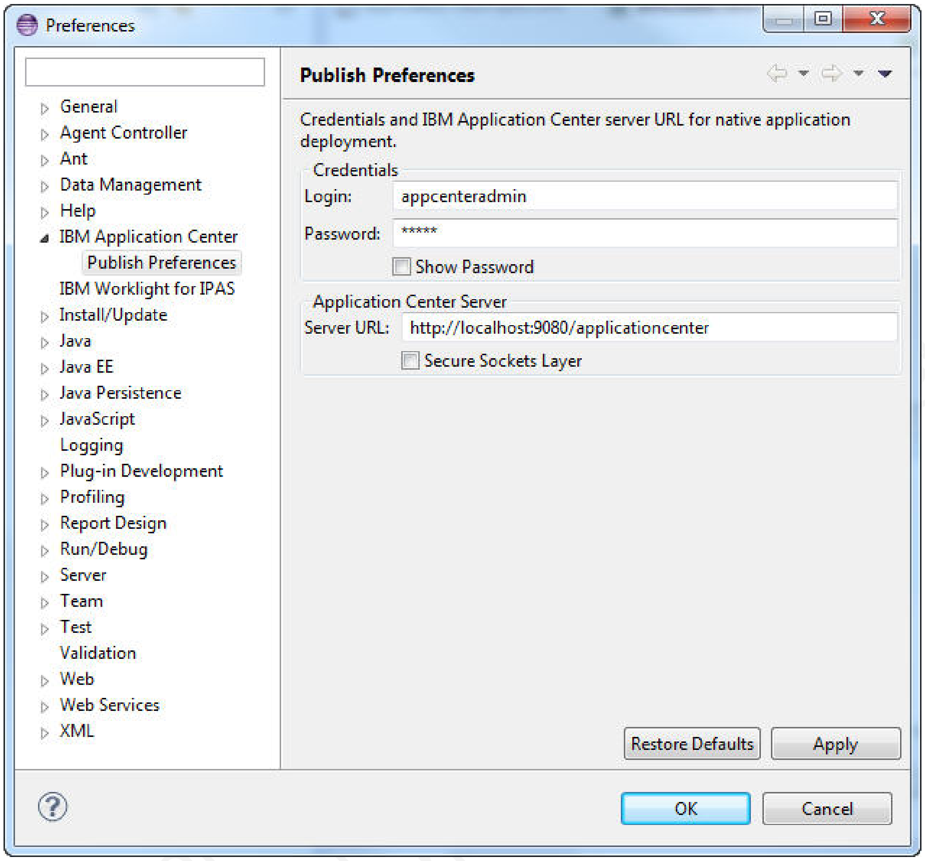
About this task

You can deploy applications for Android, 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**.



Preferences

- General
- Agent Controller
- Ant
- Data Management
- Help
- IBM Application Center
  - Publish Preferences
- IBM Worklight for IPAS
- Install/Update
- Java
- Java EE
- Java Persistence
- JavaScript
- Logging
- Plug-in Development
- Profiling
- Report Design
- Run/Debug
- Server
- Team
- Test
- Validation
- Web
- Web Services
- XML

Publish Preferences

Credentials and IBM Application Center server URL for native application deployment.

Credentials

Login: appcenteradmin

Password: *****

□ Show Password

Application Center Server

Server URL: http://localhost:9080/applicationcenter

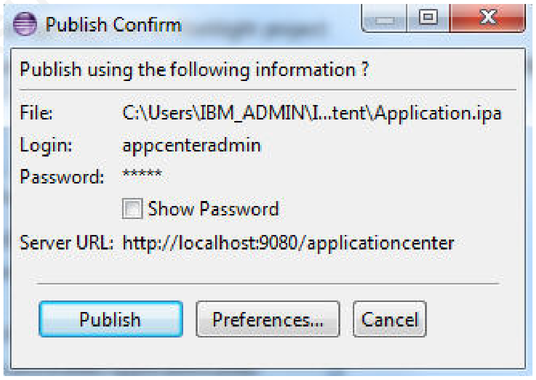
□ Secure Sockets Layer

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.



Publish Confirm

Publish using the following information ?

File: C:\Users\IBM_ADMIN\I...tent\Application.ipa

Login: appcenteradmin

Password: *****

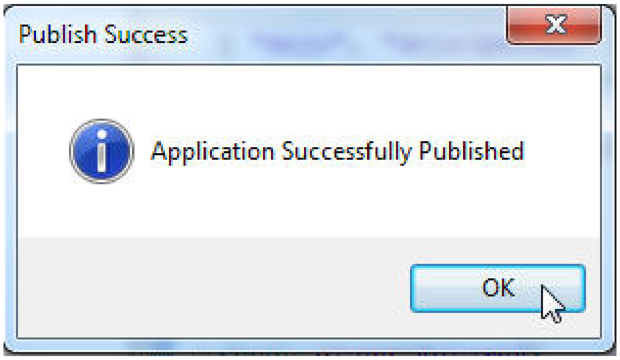
□ Show Password

Server URL: http://localhost:9080/applicationcenter

   b. In the Publish Confirm dialog, choose one of the following options:

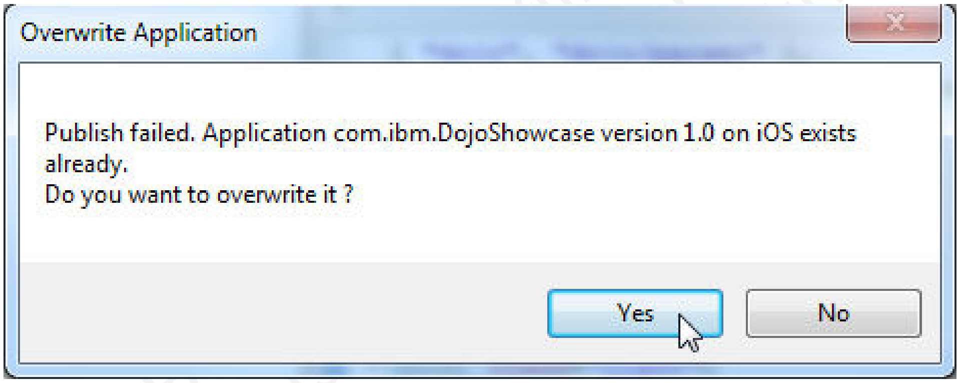
Option	Description
Publish the application by using the current preferences.	Click **Publish**.
Change any of the preferences before publishing the application.	Click **Preferences** to open the Publish Preferences page and edit the preference settings.

You receive confirmation when publication is successful.


Publish Success

Application Successfully Published

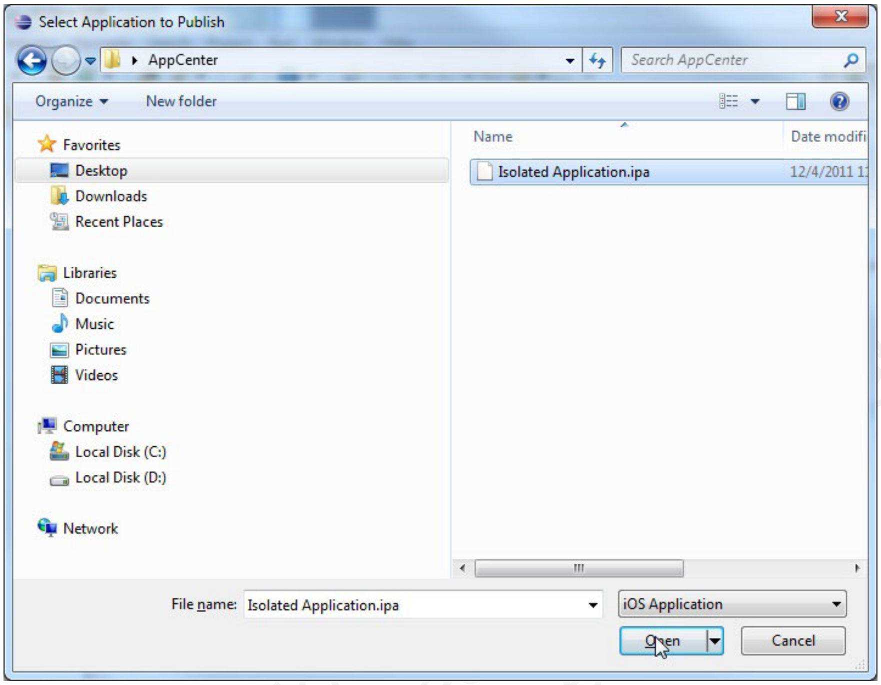
If the application already exists, publication will fail. You are given the option to overwrite the existing version of the application.


Overwrite Application

Publish failed. Application com.ibm.DojoShowcase version 1.0 on iOS exists already.
Do you want to overwrite it ?

Tip: To publish an application that is not part of the Worklight project:

1) Right-click the Worklight project and click **IBM Application Center** > **Publish on IBM Application Center**. The Select Application to Publish window opens.



Isolated Application.ipa 12/4/2011 1

File name: Isolated Application.ipa

iOS Application

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, Windows Phone, or BlackBerry device. (Only Windows Phone 8 and BlackBerry OS 6 and OS 7 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, or any signed application marked with the installer flag, 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`. This page is provided in English only and is not translated into other languages.

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 896.

   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 873.

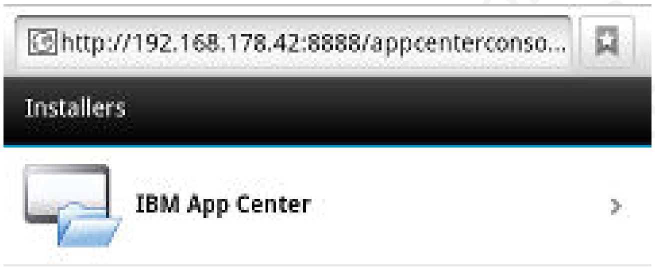
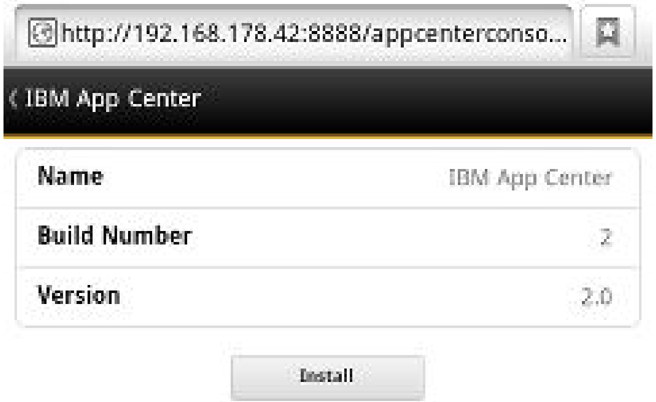


Figure 147. List of available mobile client applications to install

4. Select an item in the list to display the application details.

   Typically, these details include the application name and its version number.


http://192.168.178.42:8888/appcenterconso...

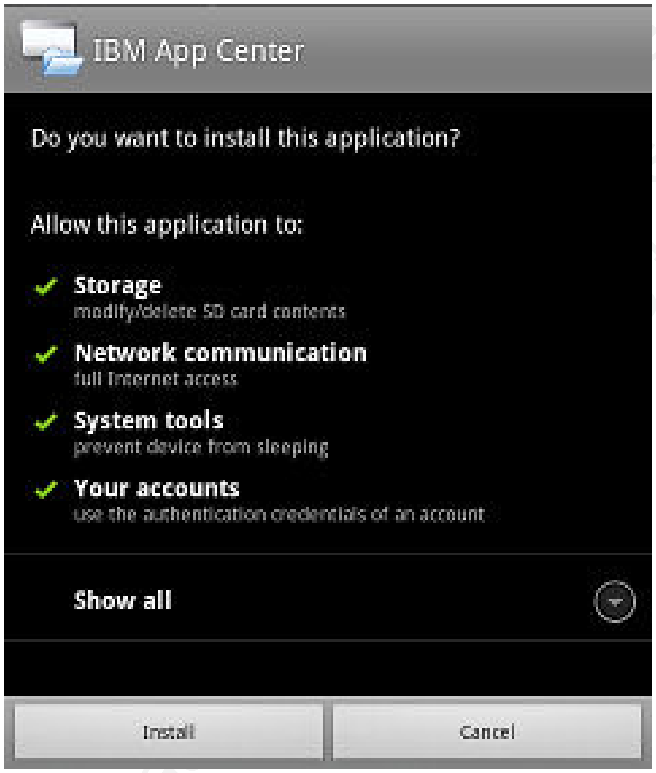
‹ IBM App Center

Name	IBM App Center
Build Number	2
Version	2.0

Figure 148. 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.


IBM App Center

Do you want to install this application?

Allow this application to:

✓ **Storage**
modify/delete SD card contents

✓ **Network communication**
full Internet access

✓ **System tools**
prevent device from sleeping

✓ **Your accounts**
use the authentication credentials of an account

Figure 149. 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, or any signed application marked with the installer flag, on your iOS mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

Before you begin

For experts

The **ibm.appcenter.ios.plist.onetimeurl** JNDI property of the IBM Application Center Services controls whether One-Time URLs are used when the mobile client is installed on an iOS mobile device. Set this property to `false` for maximal security. When you set this property to `false`, users must enter their credentials several times when they install the mobile client: once when they select the client and once when they install the client.

When you set the property to `true`, users enter their credentials only once. A temporary download URL with a cryptographic hash is generated when the user enters the credentials. This temporary download URL is valid for one hour and does not require additional authentication. This solution is a compromise between security and ergonomy.

The steps to specify the **ibm.appcenter.ios.plist.onetimeurl** JNDI property are similar to the steps for the **ibm.appcenter.proxy.host** property. See “Defining the endpoint of the application resources” on page 165.

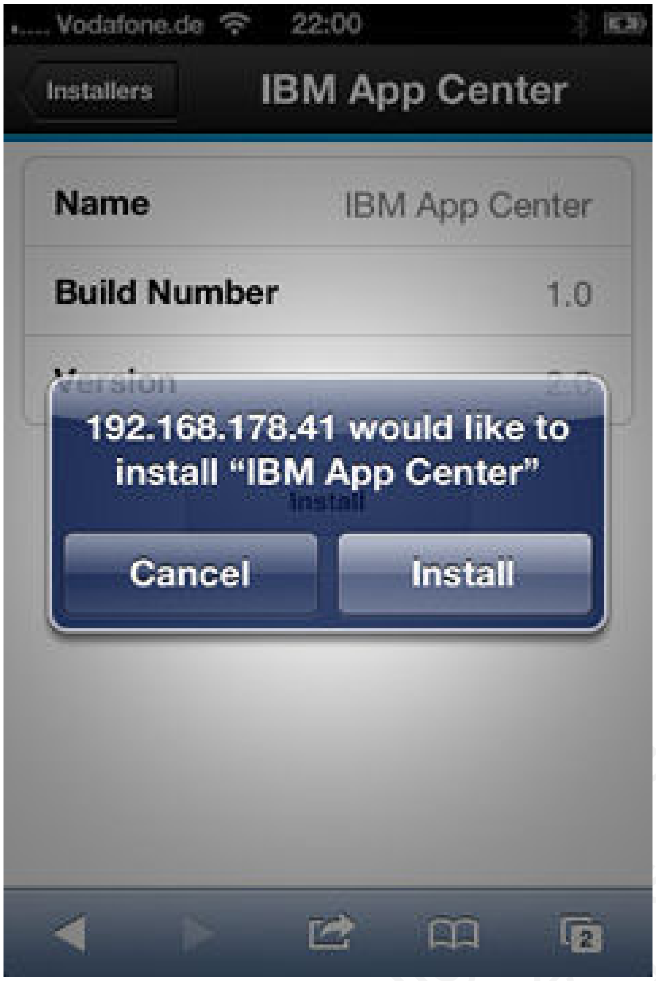
Procedure

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`. This page is provided in English only and is not translated into other languages.
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**.



Name IBM App Center

Build Number 1.0

Version 2.0

192.168.178.41 would like to install “IBM App Center”

Figure 150. 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 an application on a device requires a provisioning profile that enables the application to be installed on the selected device. If you accidentally try to install an application that is not valid for your device, iOS version 6 or earlier gives an error message. Some versions of iOS 7 might try to install the application in an endless loop without ever succeeding or indicating any error. The application icon that shows the progress of the installation appears on the home screen, but, because of the endless loop, it is difficult to delete this application icon to stop the endless loop. A workaround is to put the device into Airplane mode. In this mode, the endless loop is stopped and you can delete the application icon by following the normal steps to delete apps on iOS devices.

Installing the client on a BlackBerry mobile device

You can install the mobile client, or any signed application marked with the installer flag, on your BlackBerry mobile device by entering the access URL in your browser, entering your credentials, and completing the required steps.

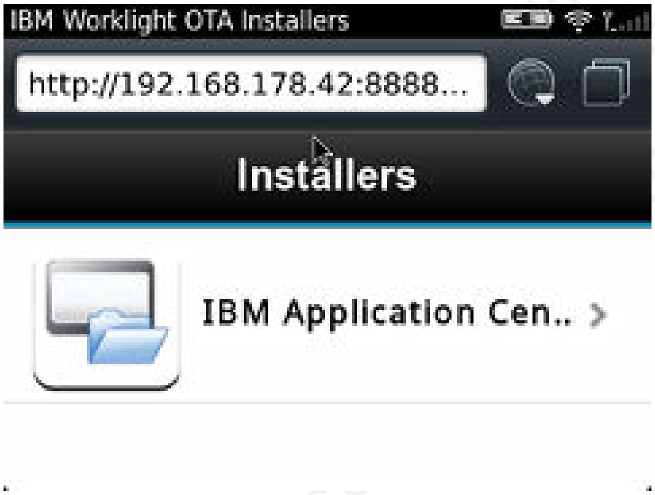
Procedure

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`. This page is provided in English only and is not translated into other languages.
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.



IBM Worklight OTA Installers

http://192.168.178.42:8888...

Installers

IBM Application Cen..

Figure 151. The installer in the BlackBerry browser

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

Installing the client on Windows Phone 8

You can install the mobile client, or any signed application marked with the installer flag, on Windows Phone 8 by entering the access URL in your browser, entering your credentials, and completing the required steps. The company account must be preinstalled on your mobile device.

Before you begin

Before you can install apps published by your company, you must add the company account to your mobile device. You must download an application enrollment token (AET) to your Windows Phone device. This AET must already be present on the Worklight Server. It is uploaded to the Worklight Server by using the Application Center console. See “Application enrollment tokens in Windows Phone 8” on page 886 for details.

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`. This page is provided in English only and is not translated into other languages.
3. Enter your credentials to authorize access to the server.

   In the lower part of the screen, a toolbar contains **Installers** tab and **Tokens** tab.

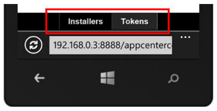
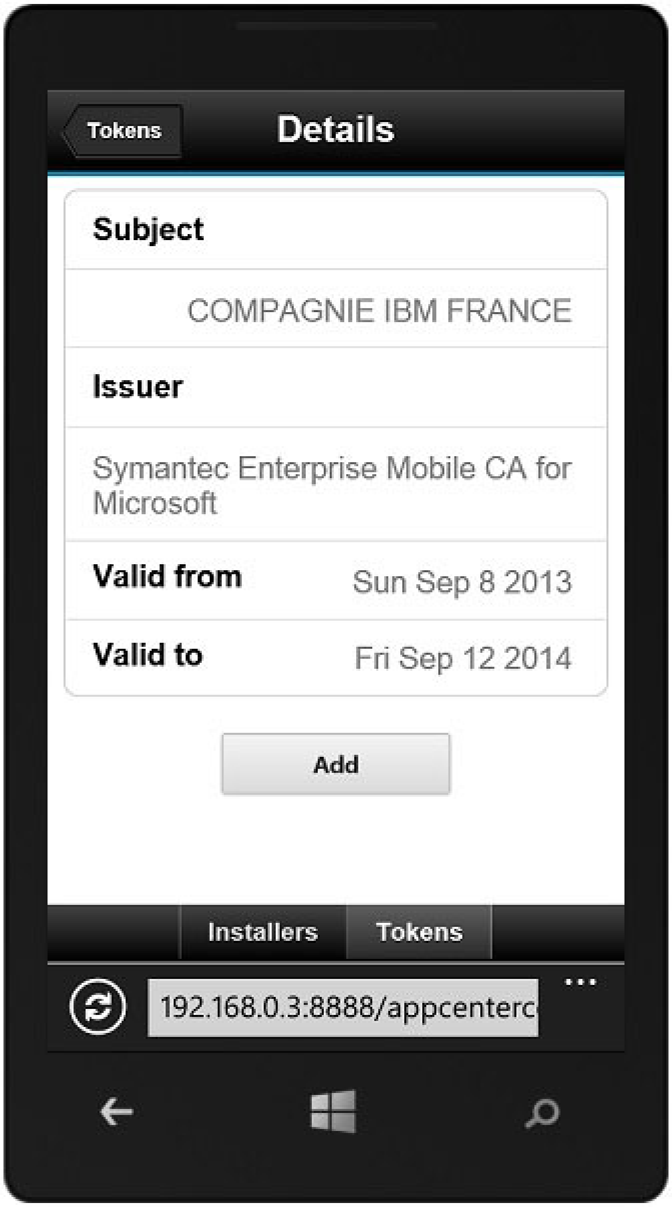


Figure 152. Preparing to install tokens and applications on a Windows Phone device

4. Tap **Tokens** and select an application enrollment token in the list of available tokens to display the token details.


Tokens

Details

Subject

COMPAGNIE IBM FRANCE

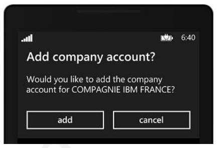
Issuer

Symantec Enterprise Mobile CA for Microsoft

Valid from	Sun Sep 8 2013
Valid to	Fri Sep 12 2014

Figure 153. AET details on a Windows Phone device

5. Tap **Add** to download the application enrollment token.
6. Tap **Add** to add the company account.



Add company account?

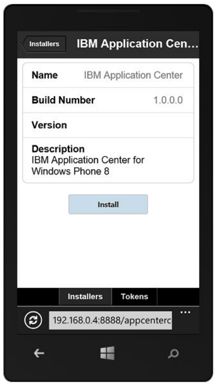
Would you like to add the company account for COMPAGNIE IBM FRANCE?

Figure 154. Adding a company account in Windows Phone 8

Windows Phone 8 does not provide any feedback about adding the company account.

7. Tap the Back icon to return to the details of application enrollment tokens.

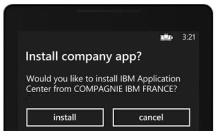
8. Tap **Installers** and select the mobile client application in the list of available applications. The application details are displayed.
9. Tap **Install** to download the selected application.



Name IBM Application Center
Build Number 1.0.0.0
Version
Description
IBM Application Center for Windows Phone 8

Figure 155. The application selected to download on a Windows Phone device

10. Tap **Install** to install the application.



Install company app?

Would you like to install IBM Application Center from COMPAGNIE IBM FRANCE?

Figure 156. Installing the downloaded application on a Windows Phone device

Windows Phone 8 does not provide any feedback about installing the application.

Tip: When you install a company application on a device, the device must connect to the Microsoft server to verify the application signature. Sometimes, the Microsoft server does not respond. In this case, you can try the installation again a few minutes later. See “Installing an application on a Windows Phone device” on page 913 for the possible error messages.

Results

When the installation is finished, the mobile client application should be available in your applications list in Windows Phone.

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.

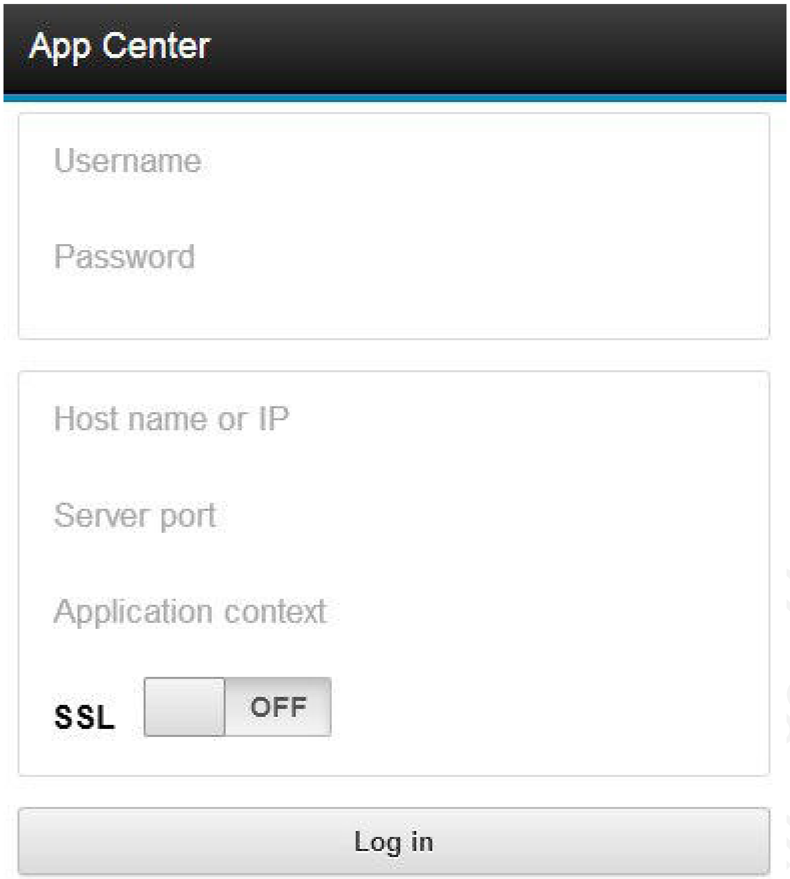
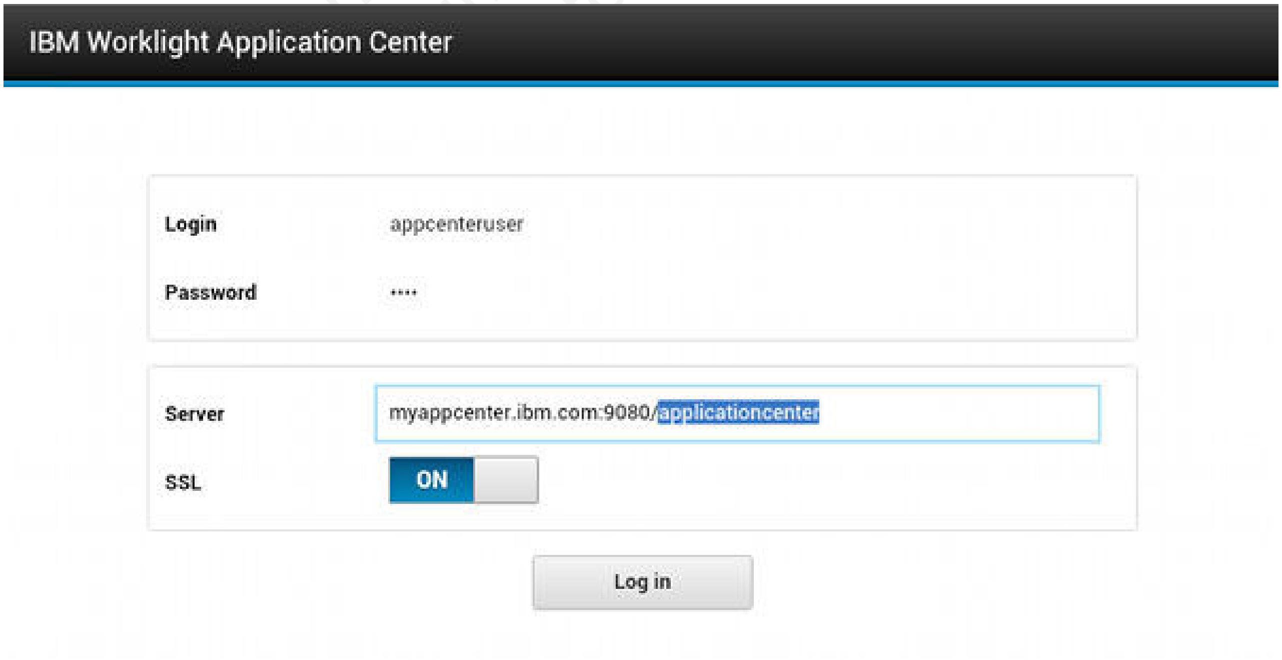


Figure 157. Login view on Android or iOS phone



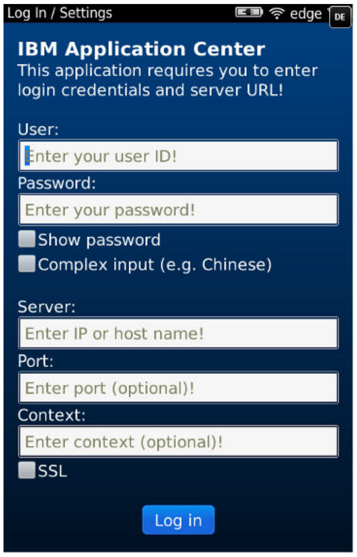
Login appcenteruser

Password ····

Server myappcenter.ibm.com:9080/applicationcenter

SSL ON

Figure 158. Login view on Android or iOS tablet



Log In / Settings edge

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 159. 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 852 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.

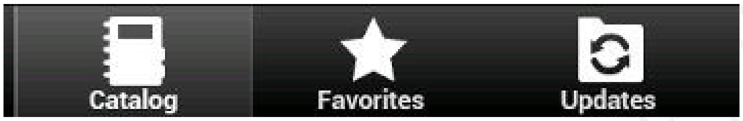


Figure 160. Views in the client application (Android and iOS operating systems)

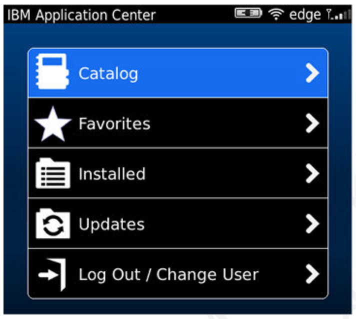


Figure 161. Views in the client application (BlackBerry devices)

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.

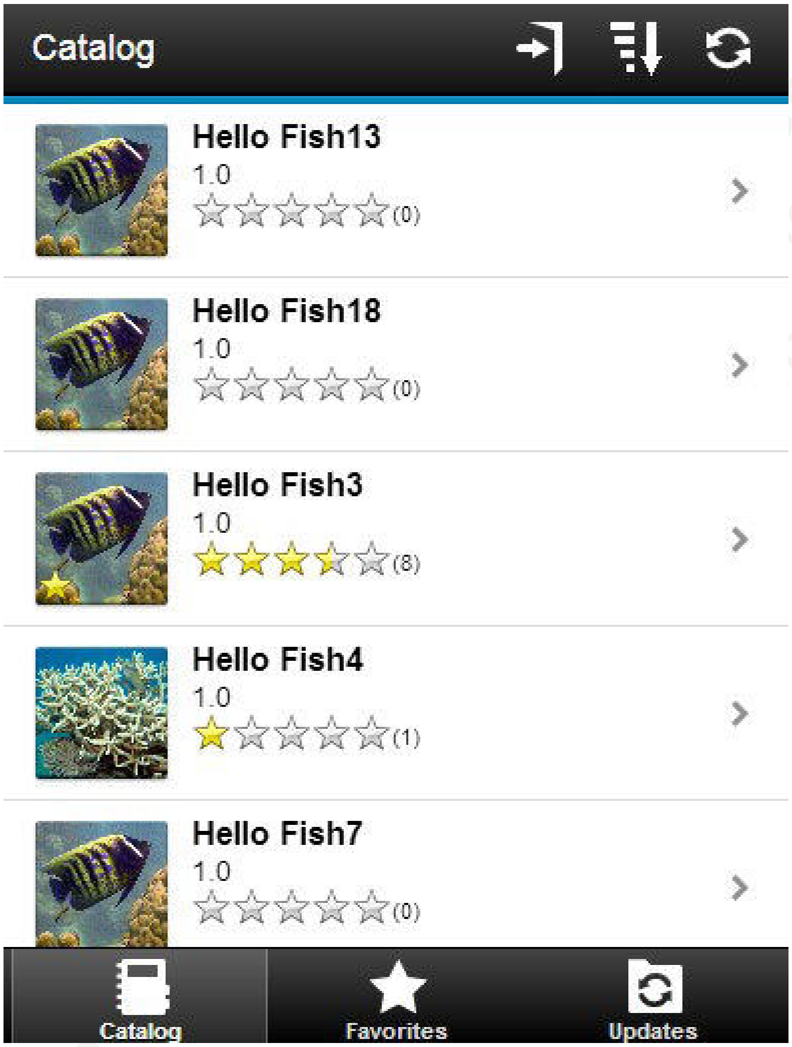


Figure 162. Catalog view on a phone

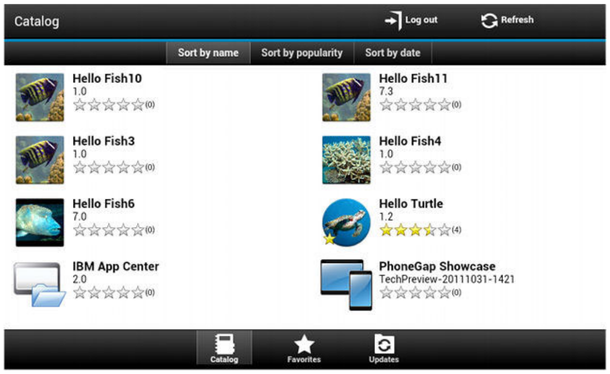


Figure 163. Catalog view on a tablet

Features of the views

On an Android or iOS tablet, you can sort the lists by tapping one of the sort criteria.

On an Android, iOS, or BlackBerry phone, 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 it 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 852 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.

To refresh the view, tap the refresh button.

To return to the login page:

- In Android and iOS applications, tap the logout button.
- In the BlackBerry version of the client, tap the return button. Then tap **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 873 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.

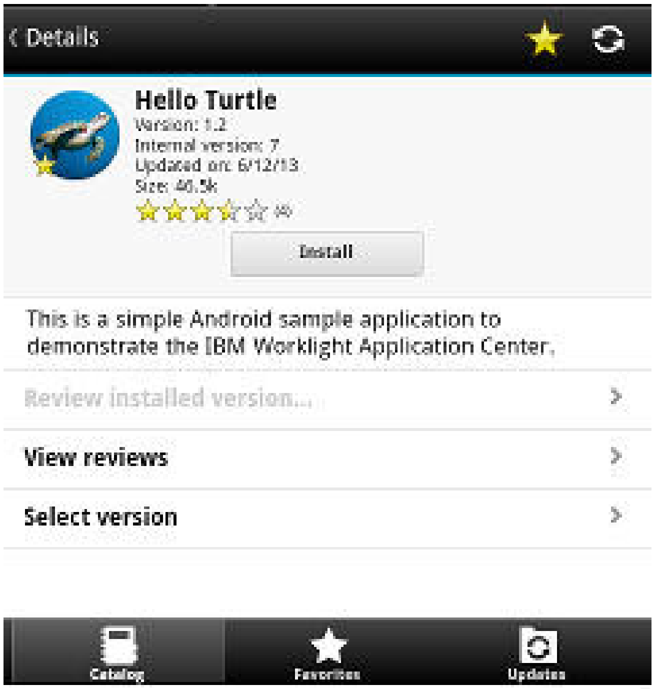
- Install, upgrade, downgrade, or uninstall an application version.
- Cancel the current operation in progress.
- Rate the application version if it is installed on the device.
- List the reviews of the this version or of all versions of the application.
- Show details of a previous version.
- Mark or 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.



Hello Turtle

Version: 1.2

Internal version: 7

Updated on: 6/12/13

Size: 46.5k

(4)

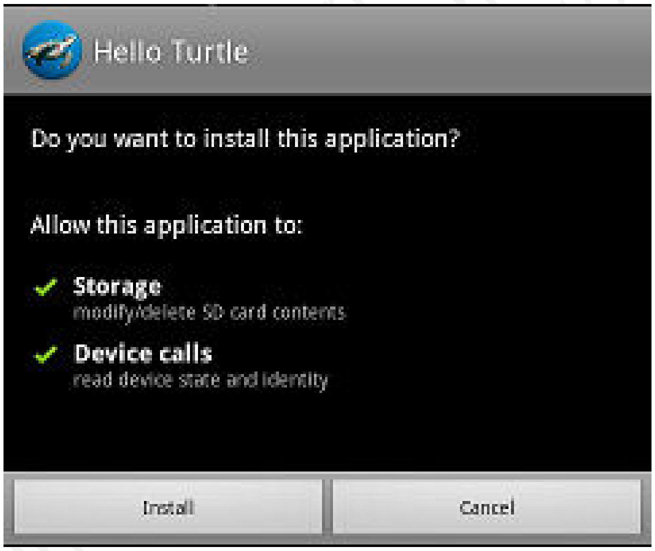
This is a simple Android sample application to demonstrate the IBM Worklight Application Center.

Figure 164. 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.



Do you want to install this application?

Allow this application to:

✓ Storage
modify/delete SD card contents

✓ Device calls
read device state and identity

Figure 165. Application rights on your Android device

2. Tap **Install** to confirm installation of the application or **Cancel** to cancel installation..

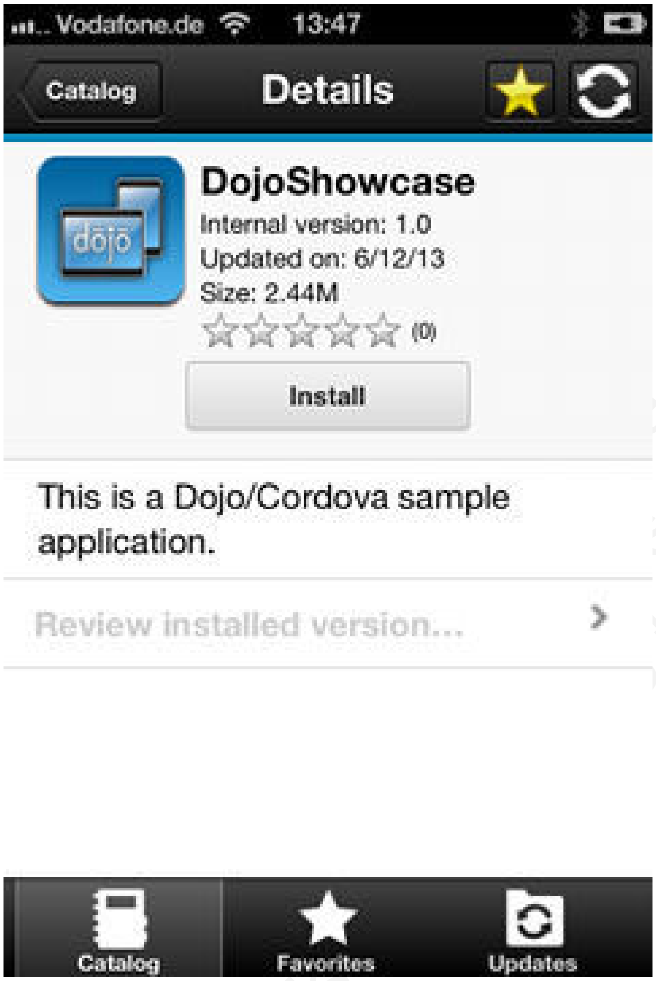
   Depending on the action taken, the application is installed or not.

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



This is a Dojo/Cordova sample application.

Figure 166. 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.

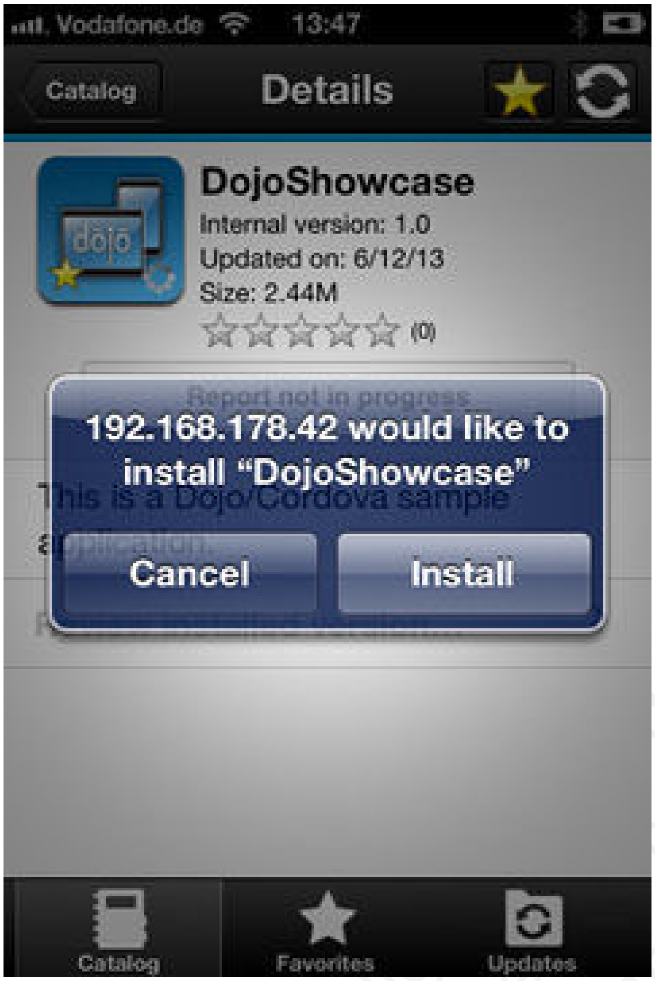


Figure 167. Canceling application installation on your iOS device

Depending on the action taken, the application is installed or not.

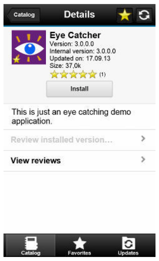
3. Optional: If you tap **Cancel**, you must notify the application that the installation has been canceled by tapping **Report Not in Progress**.

Installing an application on a Windows Phone device

From the Details view, you can install a company application on your Windows phone device.

About this task

The Details view of the selected application displays information about the application that you want to install.



Eye Catcher
Version: 3.0.0.0
Internal version: 3.0.0.0
Updated on: 17.09.13
Size: 37,0k
(1)

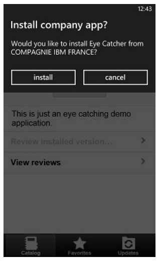
This is just an eye catching demo application.

Figure 168. Details view of a version of a company application for installation on a Windows Phone device

Procedure

1. In the **Details** view, tap **Install**. The application is downloaded and installed. You can tap **Cancel** at any time during the downloading of the application to cancel the activity. **Cancel** appears only during the downloading step of the installation process.

   At the beginning of the installation process, you are requested to confirm whether you want to add the company application to the applications installed on your mobile device.
2. Tap **Install** to confirm installation of the application or **Cancel** to cancel the installation.



Install company app?

Would you like to install Eye Catcher from COMPAGNIE IBM FRANCE?

This is just an eye catching demo application.

Figure 169. Confirming or canceling installation of a company application on a Windows Phone device

Tip: When you install a company application on a device, the device must connect to the Microsoft server to verify the application signature. Sometimes, the Microsoft server does not respond. In this case, you can try the installation again a few minutes later.

The possible error messages are:

- `There's a problem with this company app. Contact your company's support person for help.`

  You are probably using an unsigned Windows Phone application package (.xap) file. You must sign application package (.xap) files before using them in the Application Center. This message might also occur if the Microsoft server does not respond and the signature of the company application cannot be validated. In this case, try the installation again a few minutes later.
- `Before you install this app, you need to add ... company account.`

  The Windows Phone application package (.xap) file is signed, but the device is not enrolled for company applications. You must first install on the device the application enrollment token of the company.
- `We haven't been able to contact the` *`company`* `account to make sure you can install this app. ...`

Either the company account is expired or blocked, or the Microsoft server is temporarily not responding. Make sure that your device is connected to the internet and connected to the Microsoft server, and try again.

Note: If a device is registered with several company accounts, the Windows Phone operating system might display the wrong company account in the message `Would you like to install` *`application`* `from company` *`name`*`?`. This message is outside the control of the Application Center. This situation is a display problem only and does not affect the functionality.

Results

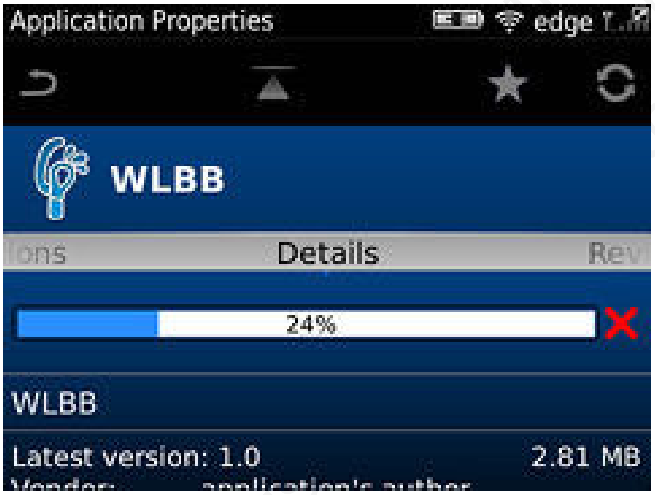
Depending on the action that you take, the application is installed or not.

Installing an application on a BlackBerry device

From the **Details** view, you can install or reinstall an application version on your BlackBerry 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. 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.



Latest version: 1.0 2.81 MB

Vendor: application's author

Figure 170. 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.

Installing applications through public app stores

You can link from the mobile client to applications that are stored in supported public app stores and install these applications on your compatible device by following the normal procedure of the public app store.

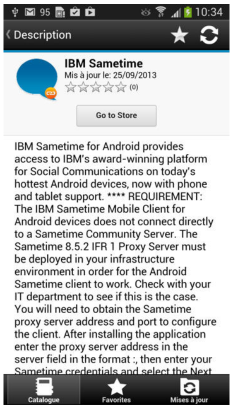
About this task

The Application Center administrator can create links to selected applications stored in supported public app stores and make them available to users of the Application Center mobile client on the operating systems that match these applications. See “Adding an application from a public app store” on page 870. You can install these applications through the mobile client on your compatible device.

Links to Android applications stored in Google play and to iOS applications stored in Apple iTunes are listed in the application list on the device along with the binary files of private applications created within your enterprise.

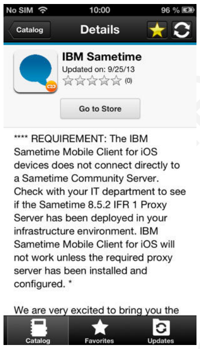
Procedure

1. Select an application stored in a public app store from the application list to see the application details. Instead of **Install**, you see **Go to Store**.
2. Tap **Go to Store** to open Google play or Apple iTunes.



IBM Sametime for Android provides access to IBM's award-winning platform for Social Communications on today's hottest Android devices, now with phone and tablet support. **** REQUIREMENT: The IBM Sametime Mobile Client for Android devices does not connect directly to a Sametime Community Server. The Sametime 8.5.2 IFR 1 Proxy Server must be deployed in your infrastructure environment in order for the Android Sametime client to work. Check with your IT department to see if this is the case. You will need to obtain the Sametime proxy server address and port to configure the client. After installing the application enter the proxy server address in the server field in the format :, then enter your Sametime credentials and select the Next

Figure 171. Accessing an application in Google play from the mobile client on the device



**** REQUIREMENT: The IBM Sametime Mobile Client for iOS devices does not connect directly to a Sametime Community Server. Check with your IT department to see if the Sametime 8.5.2 IFR 1 Proxy Server has been deployed in your infrastructure environment. IBM Sametime Mobile Client for iOS will not work unless the required proxy server has been installed and configured. *

We are very excited to bring you the

Figure 172. Accessing an application in Apple iTunes from the mobile client on the device

3. Follow the usual procedure of the public app store to install the application.

Removing an installed application

You can remove an application that is installed on your mobile device.

Procedure

1. Start the removal procedure that is valid for the operating system of your device.

- Android: See the procedure in step 2.
- iOS: You can remove applications only from the iOS Home screen, and not through the Application Center client. Use the normal iOS procedure for removing an application.
- Windows Phone: You can remove applications only from the Windows Phone Home screen, and not through the Application Center client. Use the normal Windows Phone 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 or tablet, a Windows Phone device, or a BlackBerry device.

Procedure

1. Show details of a specific application version on a mobile device by selecting the appropriate procedure to follow for your device.
   - A Windows Phone, Android, or iOS phone; see step 2.
   - A BlackBerry phone; see step 3 on page 918.
   - A tablet; see step 4 on page 918.
2. **Windows Phone, Android, iOS only**: Show details of a specific application version on a Windows Phone, Android, or iOS phone.
   a. Tap **Select a version** to navigate to the version list view.

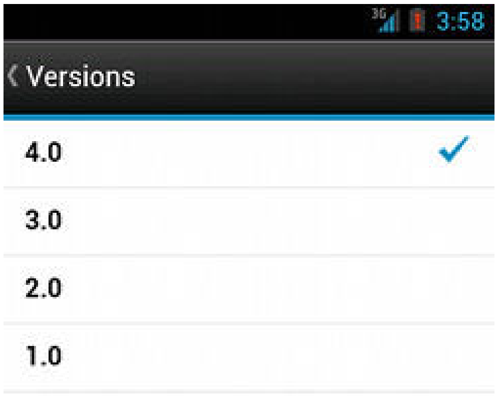


Figure 173. Specific version of an application selected in the list of versions on a Windows Phone, Android, or iOS phone

   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.

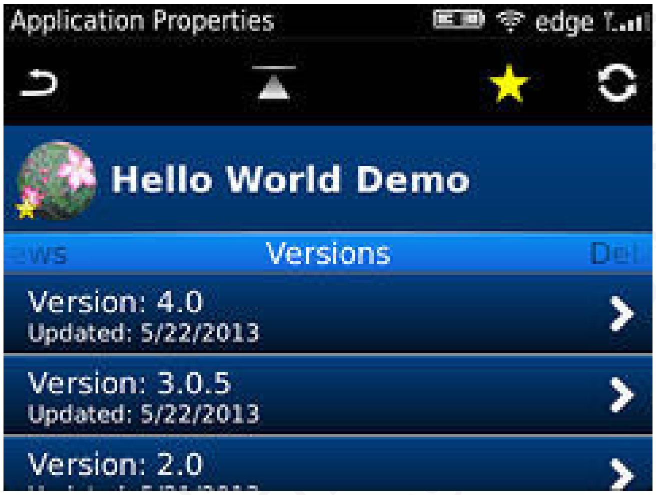


Figure 174. List of versions of an application on a BlackBerry phone

   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, Windows Phone, and BlackBerry only**: On Android, Windows Phone, 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 910
   - "Installing an application on an iOS device" on page 912
   - "Installing an application on a Windows Phone device" on page 913
   - "Installing an application on a BlackBerry device" on page 915

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 917 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 852 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 910.

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 912.

On Windows Phone

Tap **Revert**. Follow the procedue documented in "Installing an application on a Windows Phone device" on page 913.

On BlackBerry

Tap or click **Revert**. Follow the procedure documented in "Installing an application on a BlackBerry device" on page 915.

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, iOS, and Windows Phone 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 a Windows Phone, 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 a Windows Phone, 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 180. 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

Table 180. Details for manual connection (continued)

Connection suffix	User type	Conditions	Connection path
deviceside=true	All users	The mobile carrier must allow data connections. The device service book must allow direct TCP. The mobile carrier's APN must be set up.	Device > Mobile carrier > Internet > IBM Application Center Server
deviceside=true;apn=xyz		Similar to deviceside=true, but uses the specified APN.	
deviceside=true;apn=xyz;TunnelAuthUsername=user;TunnelAuthPassword=password		Similar to deviceside=true, but uses the specified APN and user name and password.	
deviceside=true;ConnectionUID=xyz	All users	The mobile carrier must allow data connections. The device service book must allow WAP 2.0. The WAP 2.0 connection details for the UID must be set up in the service book.	Device > Mobile carrier > WAP 2.0 Gateway > Internet > IBM Application Center Server

deviceside=true;WapGatewayIP=127.0.0.1;WapGatewayPort=9201;
WapGatewayAPN=xyz
	All users	The mobile carrier must allow data connections. The device service book must allow WAP 1.0/1.1.	Device > Mobile carrier > WAP 1.0 /1.1 Gateway > Internet > IBM Application Center Server

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.	

Table 180. Details for manual connection (continued)

Connection suffix	User type	Conditions	Connection path
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

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 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. For options to use FIPS 140-2 validated libraries on client devices, see “FIPS 140-2 on the IBM Worklight client device for protection of data at rest in JSONStore and data in motion when using HTTPS communications.”
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 “References” on page 927 for links to documentation to enable FIPS 140-2 mode in WebSphere Application Server.

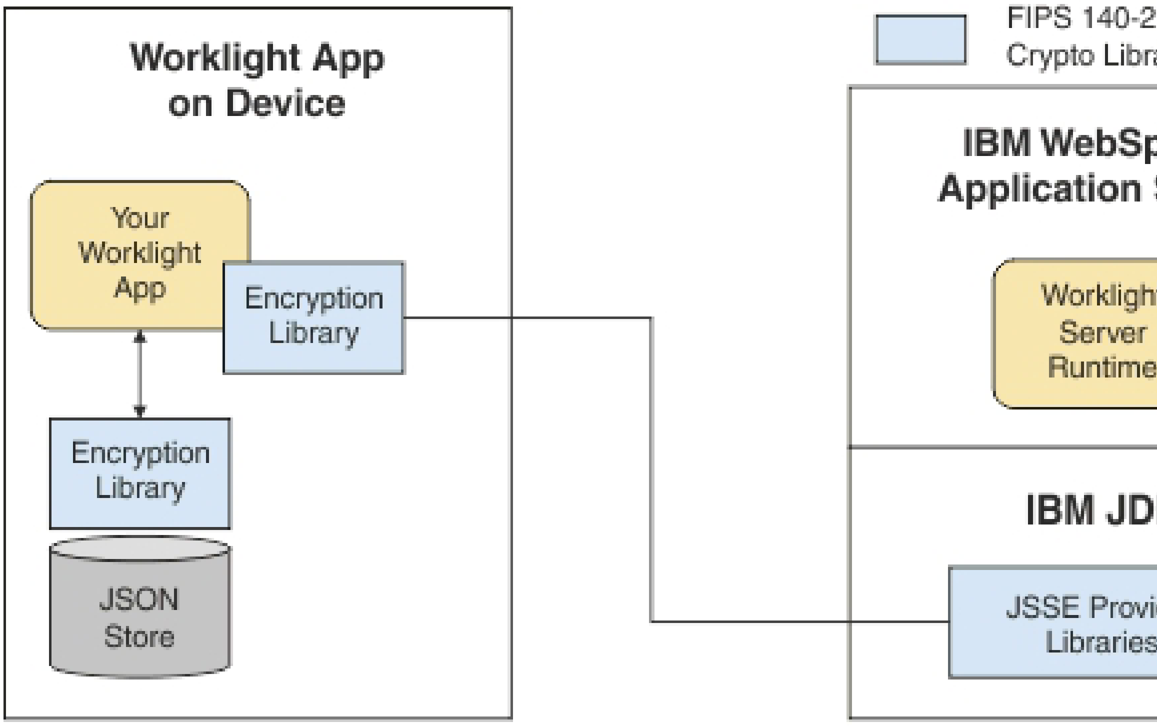
FIPS 140-2 on the IBM Worklight client device for protection of data at rest in JSONStore and data in motion when using HTTPS communications

Protection of data at rest on the client device is provided by the JSONStore feature of IBM Worklight. Protection of data in motion is provided by the use of HTTPS communication between the Worklight client and the Worklight Server. By default, these features use 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 and for the HTTPS communication to the Worklight Server. This support is achieved by using an OpenSSL library that achieved FIPS 140-2 validation (Certificate #1747). To enable this option in a Worklight client project, select the FIPS 140-2 optional feature in the Worklight Studio.

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 and protection of HTTPS communications between the Worklight client and the Worklight Server.
- 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.
- On iOS, it is supported on **`i386`**, **`armv7`**, and **`armv7s`** architectures.
- It only works with hybrid applications (not native).
- For HTTPS communications, only the communications between the Worklight client and the Worklight Server use the FIPS 140-2 libraries on the client. Direct connections to other servers or services do not use the FIPS 140-2 libraries.
- The use of the analytics feature (Tealeaf) on the client is not supported with the FIPS 140-2 feature.
- The use of the User Certificate Authentication feature is not supported with the FIPS 140-2 feature.
- The Worklight Application Center client does not support the FIPS 140-2 feature.
- The use of the Direct Update feature is not supported by the FIPS 140-2 feature.

The FIPS 140-2 optional feature supersedes the changes that were described in the module *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for previous versions of IBM Worklight under category 5, *Advanced client side development*. If you previously made the changes that are described in this module, you must delete and recreate your Android or iOS environments after first saving any other environment-specific changes that you made.

Figure 175. Example



For more information about JSONStore, see "JSONStore overview" on page 567.

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.

Enabling FIPS 140-2

To use the FIPS 140-2 feature in IBM Worklight V6.1.0, you must first enable the optional feature.

About this task

To enable this option in a Worklight client project, select the FIPS 140-2 optional feature in the Worklight Studio. After the optional feature is enabled, it must then be configured as described in the *What to do next* section. After the FIPS 140-2 optional feature is enabled and configured, this feature applies both to HTTPS and JSONStore data encryption.

Note: FIPS 140-2 is only supported on Android and iOS, and only for **`i386`**, **`armv7`**, and **`armv7s`** architectures.

The FIPS 140-2 feature is an optional feature in IBM Worklight V6.1.0. To use the FIPS 140-2 feature, you must enable it by modifying the `application-descriptor.xml` file.

Procedure

1. Double-click the `application-descriptor.xml` file to open it in the Application Descriptor Editor.
2. Click the **Design** tab.
3. Under **Overview**, expand `Application [your application's name]`.
4. Click **Optional Features**.
5. Click **Add**.
6. Select **FIPS 140-2**.
7. Click **OK**.

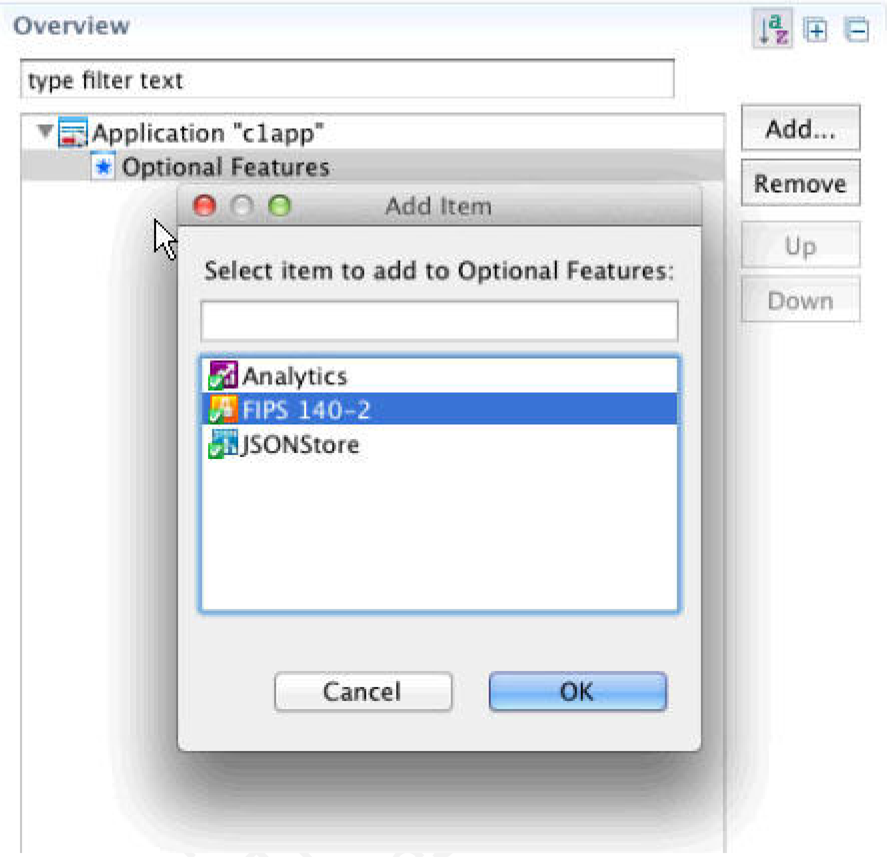


Figure 176. Installing the FIPS 140-2 optional feature

What to do next

"Configure FIPS 140-2 mode for HTTPS and JSONStore encryption"

Configure FIPS 140-2 mode for HTTPS and JSONStore encryption

Learn about settings to configure FIPS 140-2 for encrypting data for HTTPS and JSONStore.

The following code snippet is populated into a new IBM Worklight application in the `initOptions.js` file for configuring FIPS 140-2:

```
var wlInitOptions = {
  ...
  // # Enable FIPS 140-2 for data-in-motion (network) and data-at-rest (JSONStore) on iOS or Android
  //   Requires the FIPS 140-2 optional feature to be enabled also.
  // enableFIPS : false
  ...
};
```

Notice the default value of `enableFIPS` is false. To enable FIPS 140-2 for both HTTPS and JSONStore data encryption, uncomment and set the option to `true`. After you set the value of `enableFIPS` to true, you should listen for the FIPS ready JavaScript event.

The following example assumes that you are using `jQuery 1.7` or later, or `WLJQ` (`jQuery` that is included with IBM Worklight).

```
$(document).on('WL/FIPS/READY', function(evt, obj) {
  //evt - Contains information about the event
  //obj - JavaScript object sent after the attempt to enable FIPS completes
  // if successfully enabled, object will be {enabled: true}
  // if enablement failed, object will be {enabled: false, msg: "message
  // indicating cause of the failure to enable"}
});
```

After you set the value of the `enableFIPS` property, create an Android, iPhone, or iPad environment, and build those environments.

Note: You must enable the FIPS 140-2 optional feature before you set the `enableFIPS` property to `true`. Otherwise, a warning message is logged that states the `initOption` value is set but the optional feature was not found. The FIPS 140-2 and JSONStore features are both optional. FIPS 140-2 affects JSONStore data encryption only if the JSONStore optional feature is also enabled. If JSONStore is not enabled, then FIPS 140-2 does not affect JSONStore.

```
[WARN] FIPSHttp feature not found, but initOptions enables it on startup
```

For more information about installing the FIPS 140-2 optional feature, see “Enabling FIPS 140-2” on page 927.

Configuring FIPS 140-2 for existing applications

You must modify applications that were created in earlier versions of IBM Worklight to enable the FIPS 140-2 feature.

Before you begin

The FIPS 140-2 optional feature is not enabled by default. To enable the FIPS 140-2 optional feature, see “Enabling FIPS 140-2” on page 927. After the optional feature is enabled, you can configure FIPS 140-2.

About this task

After you completed the steps that are described in “Enabling FIPS 140-2” on page 927, you must configure FIPS 140-2 by modifying the `initOptions.js` file to add the FIPS configuration property.

Note: For JSONStore FIPS 140-2 users – Starting with IBM Worklight V6.1.0, the FIPS 140-2 feature, combined with the JSONStore feature, enables FIPS 140-2 support for JSONStore. This combination supersedes what was detailed in the module *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for previous versions of IBM Worklight under category 5, *Advanced client side development*. If you previously modified an application by following the instructions in this module, delete and re-create its iPhone, iPad, and Android environments. Because any environment-specific changes that you previously made are lost when you delete an environment, make sure to back up any such changes before you delete any environment. After the environment is re-created, you can reapply those changes to the new environment.

Procedure

1. Add the following lines of code to the `initOptions` object found in the `[appFolder]/common/js/initOptions.js` file.

   ```
   enableFIPS : true
   ```
2. Rebuild and deploy your app.

Avoiding an archive build failure with the FIPS 140-2 optional feature on iOS

You can avoid an archive build failure with the FIPS 140-2 optional feature on iOS by changing the Xcode linker options.

About this task

When the FIPS 140-2 optional feature is included in a Worklight Studio project with an iPhone or iPad environment, you must change the Xcode linker options before you archive the product (for example, if you want to create an IPA file).

If you do not change the linker options, the archive build fails with an error that is similar to the following text:

```
4196 duplicate symbols for architecture armv7
Linker command failed with exit code 1 (use -v to see
invocation)
```

To resolve the problem, change the linker options in Xcode by completing the following steps:

Procedure

1. Select the **Build Settings** for your project.
2. Search for `-all_load`. You should find two occurrences under "Other Linker Flags": one is for Distribution and another one is for Release.
3. Double-click the **Distribution** value to open the editing window.
   a. Click **+** on the lower left corner of the window and add the following option:

      ```
      -force_load $(BUILT_PRODUCTS_DIR)/libCordova.a
      ```
   b. Click **+** on the lower left corner of the window and add the following option:

      ```
      -force_load $(SRCROOT)/WorklightSDK/libWorklightStaticLibProject.a
      ```
   c. Click **+** on the lower left corner of the window and add the following option:

      ```
      -force_load $(SRCROOT)/FipsHttp/libfipshttp.a
      ```
   d. Select the `-all_load` line and remove it by clicking **-** on the lower left corner of the window.
   e. Click outside of the editing window to save the changes and close this window.
4. Repeat these steps for the **Release** value.

Results

After you change the Xcode linker options, the archive build should succeed.

Chapter 12. Monitoring and mobile operations

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 181. Documentation for different server platforms

Server platform	Location of documentation
Apache Tomcat	http://tomcat.apache.org/tomcat-7.0-doc/logging.html#Using_java.util.logging_(default)
WebSphere Application Server Version 7.0	http://pic.dhe.ibm.com/infocenter/wasinfo/v7r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.0	http://pic.dhe.ibm.com/infocenter/wasinfo/v8r0/topic/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 Full Profile	http://pic.dhe.ibm.com/infocenter/wasinfo/v8r5/topic/com.ibm.websphere.nd.multiplatform.doc/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 Liberty Profile	http://publib.boulder.ibm.com/infocenter/radhelp/v8r5/topic/com.ibm.websphere.wlp.nd.multiplatform.doc/ae/rwlp_logging.html

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 182. Examples of queries that help determine server vitality

Query	Purpose
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality`	Checks the server as a whole.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp`	Checks the server and the `MyApp` application.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp&adapter=MyAdapter`	Checks the server, the `MyApp` application, and the `MyAdapter` adapter.

Note: Do not include the `/<publicWorkLightContext>` part of the URL if you use IBM Worklight Developer Edition. You must add this part of the URL only if 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 183. 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. Since Worklight Studio V6.0.0, the test server is replaced with an instance of the WebSphere Application Server Liberty Profile server, and is now 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

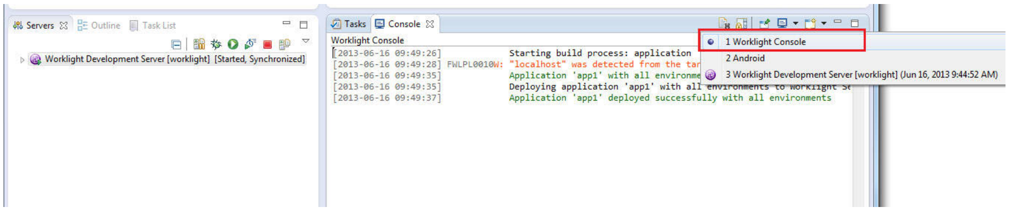
To change the logging level for all packages in this instance of Eclipse, edit the `.level =` line. To change the logging level only for Worklight Studio, edit the `com.worklight.level =` line.

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:


Servers
Outline
Task List
Worklight Development Server [worklight] [Started, Synchronized]
Tasks
Console
1 Worklight Console
2 Android
3 Worklight Development Server [worklight] (Jun 16, 2013 9:44:52 AM)

```
Worklight Console
[2013-06-16 09:49:26]           Starting build process: application
[2013-06-16 09:49:28] FWLPL0010W: "localhost" was detected from the tar
[2013-06-16 09:49:35]           Application 'app1' with all environme
[2013-06-16 09:49:35]           Deploying application 'app1' with all environments to Worklight Se
[2013-06-16 09:49:37]           Application 'app1' deployed successfully with all environments
```

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" />
  ```

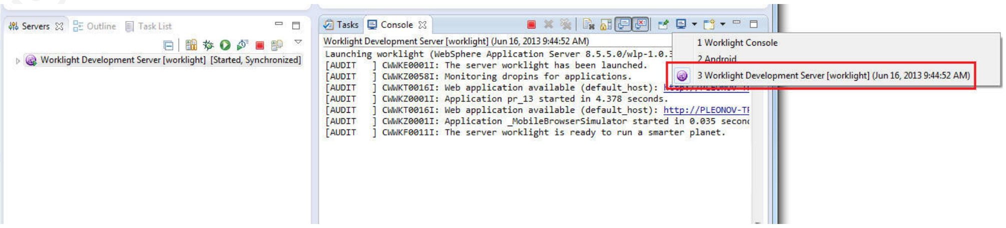
The available setting levels for `consoleLogLevel` are:

- `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:


Servers
Outline
Task List
Worklight Development Server [worklight] [Started, Synchronized]
Tasks
Console
1 Worklight Console
2 Android
3 Worklight Development Server [worklight] (Jun 16, 2013 9:44:52 AM)

```
Worklight Development Server [worklight] (Jun 16, 2013 9:44:52 AM)
Launching worklight (WebSphere Application Server 8.5.5.0/wlp-1.0.3
[AUDIT   ] CWWKE0001I: The server worklight has been launched.
[AUDIT   ] CWWKZ0058I: Monitoring dropins for applications.
[AUDIT   ] CWWKT0016I: Web application available (default_host):
[AUDIT   ] CWWKZ0001I: Application pr_13 started in 4.378 seconds.
[AUDIT   ] CWWKT0016I: Web application available (default_host): http://PLEONOV-TI
[AUDIT   ] CWWKZ0001I: Application _MobileBrowserSimulator started in 0.035 second
[AUDIT   ] CWWKF0011I: The server worklight is ready to run a smarter planet.
```

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 `servers`view.

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.

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 feature that is 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. 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 939.

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.

Dashboard view

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 Logs view

A scrolling view of the server-side log information in a table form with ability to filter the view by keywords.

Geo Analytics 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, IBM SmartCloud Analytics Embedded 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 IBM SmartCloud Analytics Embedded.

Related tasks:

Installing and configuring IBM SmartCloud Analytics Embedded
To run the analytics features, you must install IBM SmartCloud Analytics Embedded.

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 database" on page 967 feature is a subset of the total data that is collected as part of the Operational "Analytics" on page 938 feature. You can use the reports database and the operational analytics feature simultaneously, but usage of both in a production environment is redundant. Use the reports feature in cases where you want direct access to the Reports 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 analytics processing (OLAP) system that runs database queries directly against the Reports database.

Table 184. Comparison of analytics and reports features

	Operational analytics feature	Reports feature
Primary usage	Problem determination, device usage summary, geographic view of mobile activity	Device usage summary
Typical user	Administrator, operational support personnel, developer, analyst	Administrator, analyst
Data used in analytics	App crash from clients, Worklight server log, Worklight app to server interaction activities	Worklight app to server interaction activities
Data storage mechanism	Files on IBM SmartCloud Analytics Embedded	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 IBM SmartCloud Analytics Embedded.

IBM Worklight analytics components

IBM SmartCloud Analytics Embedded and several libraries make up the components of the IBM Worklight operational analytics feature.

IBM Worklight includes IBM SmartCloud Analytics Embedded, which is the engine that drives the IBM Worklight scalable operational analytics feature. This component 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.

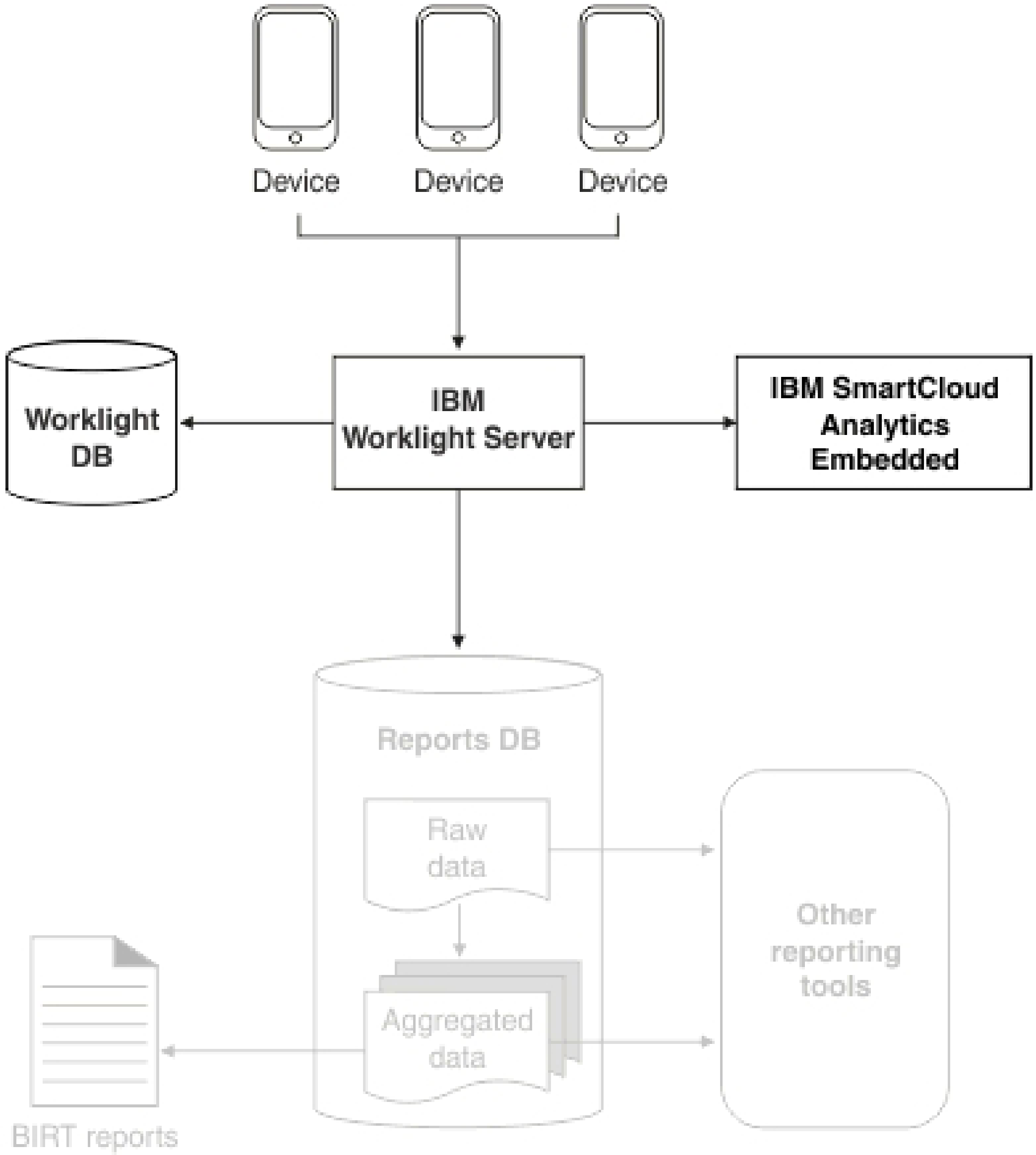


Figure 177. High-level overview of reporting and analytics infrastructure.

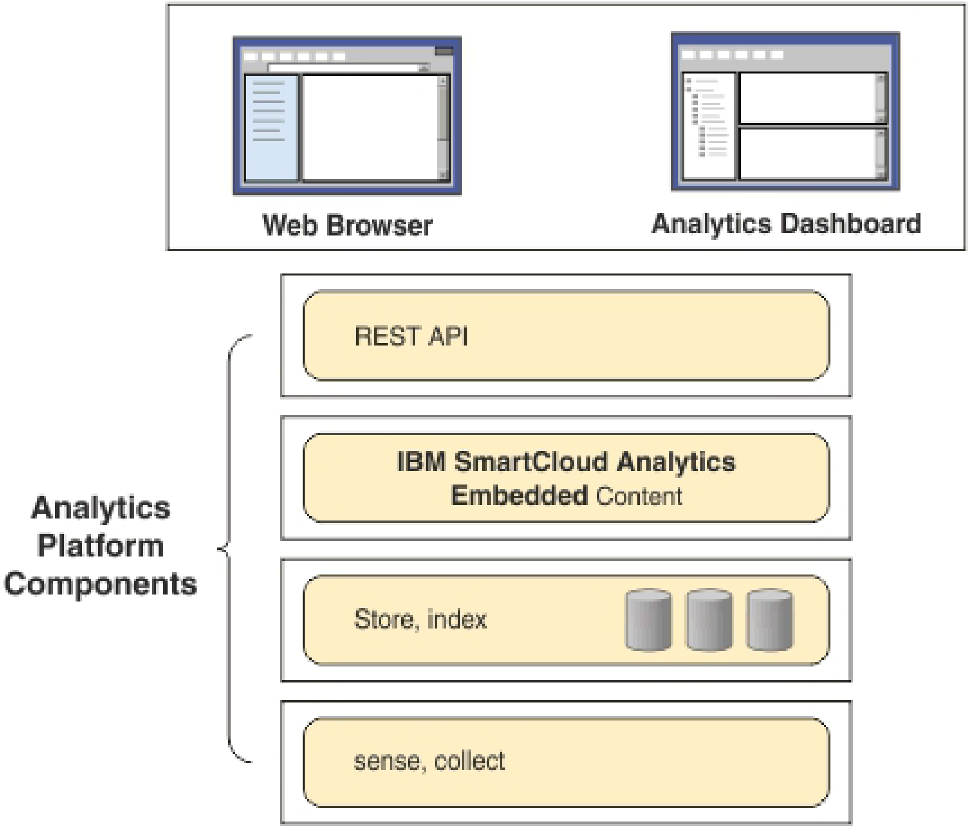
Note: Enabling reports, using BIRT to query the reports database, and enabling IBM SmartCloud Analytics Embedded are each optional.

IBM SmartCloud Analytics Embedded

IBM SmartCloud Analytics Embedded provides a real-time, extensible platform for sensing, collecting, storing, indexing, analyzing, searching, and visualizing IT and Business Data. IBM SmartCloud Analytics Embedded is built for handling large volumes of data, which can dynamically grow in scenarios like mobile traffic. IBM Worklight production environments for the enterprise typically deals with these types of scenarios.

Note: IBM SmartCloud Analytics Embedded is also referred to as the IBM WebSphere Analytics Platform.

IBM SmartCloud Analytics Embedded is customized to intake data that is generated by the IBM Worklight Servers and devices to provide specific mobile operational insights. The following figure shows a high-level architecture of IBM SmartCloud Analytics Embedded:


Web Browser
Analytics Dashboard
Analytics Platform Components
REST API
IBM SmartCloud Analytics Embedded Content
Store, index
sense, collect

At the core, IBM SmartCloud Analytics Embedded enables sensing and collection of IBM Worklight Server-specific data along with device session data by using IBM Tealeaf libraries. This data is indexed into a set of search nodes, which form the search cluster. Search cluster is elastically scalable in a way to address massively increasing amounts of mobile data that is handled by IBM Worklight.

Analytics Content is a custom designed module (also referred to as a plug-in) that understands the structure of the data that is collected from the Worklight Server and mobile devices to provide dashboards and user interfaces for the platform. IBM SmartCloud Analytics Embedded is built on Representational State Transfer (REST) APIs for individual functions like indexing, searching, visualization, and dash-boarding.

From a physical deployment standpoint, IBM SmartCloud Analytics Embedded consists of two types of nodes: Search Engine Nodes & Processing Console Nodes.

Search Engine Nodes

A search-engine node runs a search process usually running in its own physical computer. The search node can store, index, and search by using free formed text and term search, including real-time facets of different fields that are used to represent the Worklight Server and device data.

Processing Console Node

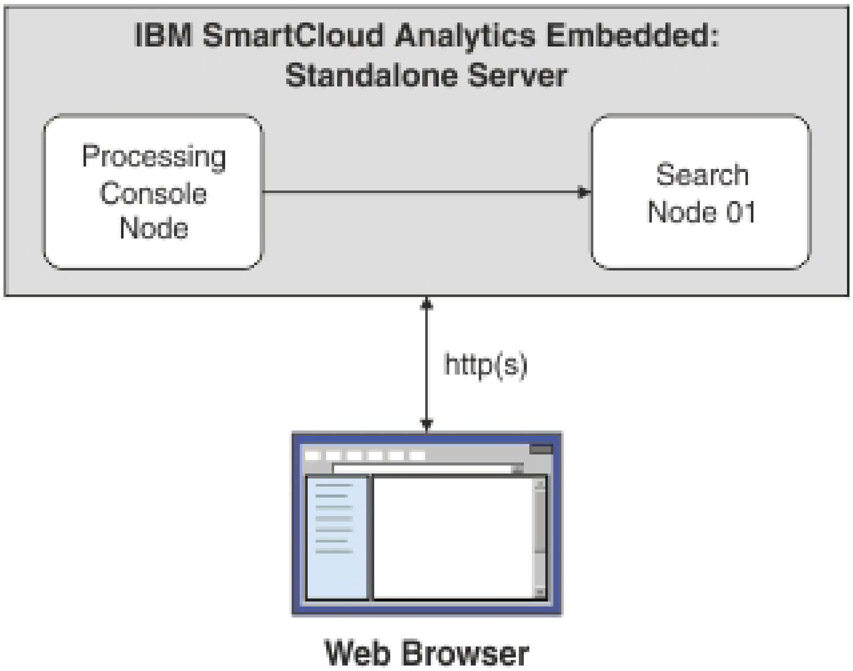
A processing console node is an HTTP Server node that provides the UI and API access to Worklight Analytics Content to users. This server is shown to users and is responsible for providing the analytics dashboard and APIs through the web browser.

IBM SmartCloud Analytics Embedded topologies:

IBM SmartCloud Analytics Embedded provides a scalable solution for real time mobile analytics. You can learn how the search and console nodes of the platform can be configured for highly scalable architecture.

Stand-alone analytics topology

IBM SmartCloud Analytics Embedded is made of primarily two types of nodes. While they are self-contained independent nodes in terms of functionality, the search and the processing console can be run in a stand-alone single server setup. The following image shows a stand-alone server topology:



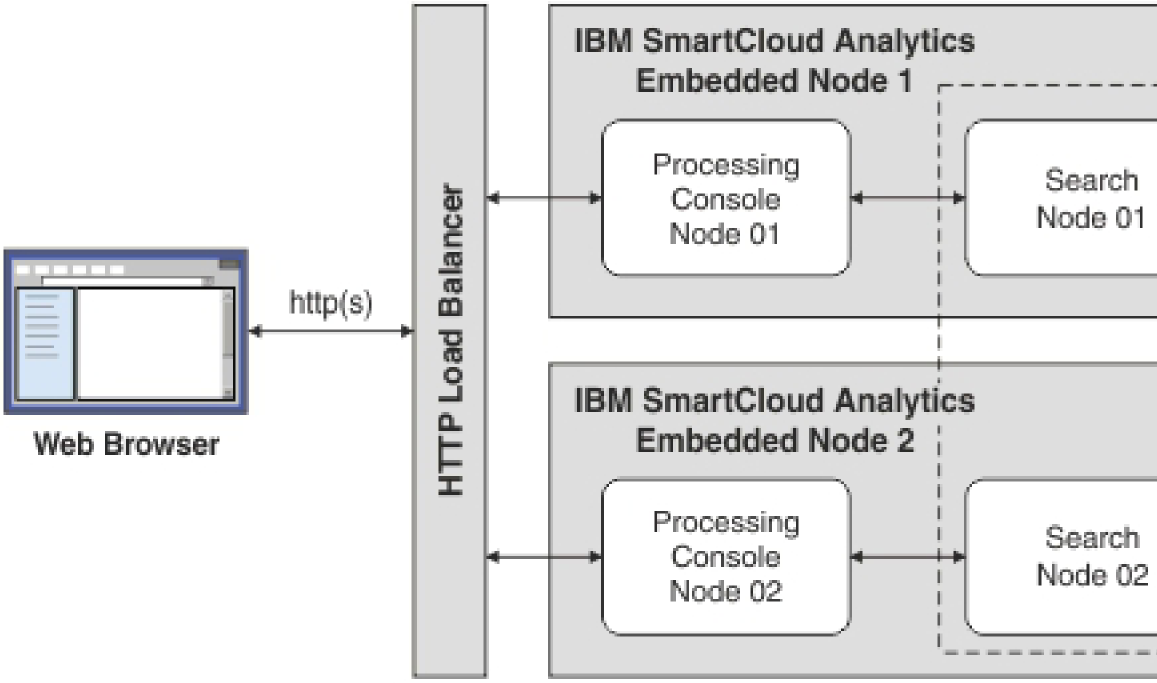
Both the processing console node and search node are installed and configured in a single server. The search cluster has a single node and stores all the data on the disk that is associated within this server.

This type of configuration is typically suitable for proof of concepts or quickly bringing up a test analytics server for viability assessment and debugging.

Scalable stand-alone analytics topology

In this type of deployment, both the processing console node and the search node are installed in a single server as a stand-alone configuration. Adding more than one stand-alone server, with both console and search nodes on a single server, can form a scalable stand-alone analytics topology.

An HTTP load balancer in the front of this topology ensures that a higher number of concurrent uses is supported equally across the two stand-alone servers. The following image shows a scalable stand-alone analytics topology:


Web Browser
http(s)
HTTP Load Balancer
IBM SmartCloud Analytics Embedded Node 1
Processing Console Node 01
Search Node 01
IBM SmartCloud Analytics Embedded Node 2
Processing Console Node 02
Search Node 02

If one of the console node processes fails, IBM SmartCloud Analytics Embedded is still functional for end users.

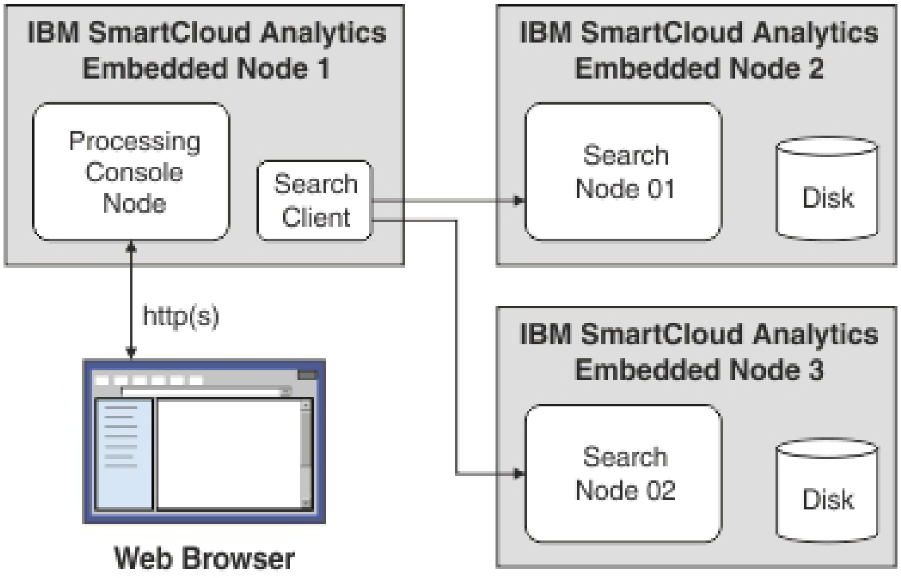
If one of the search node processes fails, the secondary search node becomes the primary node. The data is recovered on the secondary server from the replicas that are maintained in the secondary node.

The drawback with this type of deployment is that the console node that is associated with a failed search node becomes non-functional and might throw exceptions. The advantage to this type of deployment is that the computing power of the single server is shared between the processing console node and the search node.

To achieve higher performance with high availability and a highly scalable deployment, choose the highly available analytics topology.

Search optimized analytics topology

In this type of deployment, search cluster is distributed in a way that each search node runs in a separate physical server. This deployment model provides high availability and failover to the data stored by the search cluster. Data is stored in a logical container that is called index, which is replicated and stored across different search nodes that participate in the search cluster. The following image shows a search optimized topology:


IBM SmartCloud Analytics Embedded Node 1
Processing Console Node
Search Client
IBM SmartCloud Analytics Embedded Node 2
Search Node 01
Disk
http(s)
IBM SmartCloud Analytics Embedded Node 3
Search Node 02
Disk
Web Browser

The search cluster is distributed into two separate nodes that consist of individual disk mounts: disk 1 and disk 2. They are in a clustered mode, so one is selected as a primary search node and the other is selected as a secondary search node.

If a search node process goes down, the secondary node comes up as a primary node. The data is recovered on the secondary server from the replicas that are maintained in the secondary node.

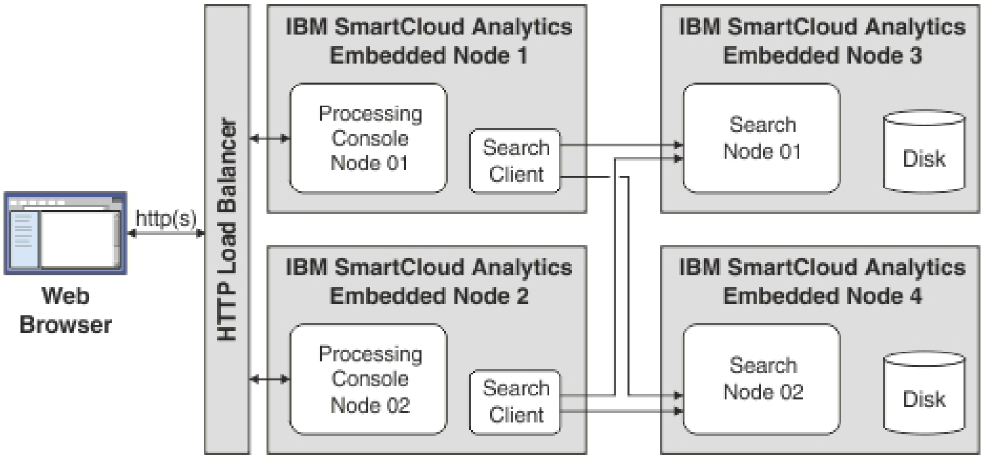
The total available space for storing Worklight Server and device data is the sum of the space available on disk 1 and disk 2. The disks and their performance play a key role in search cluster's overall performance. These disks can be mounted folders from SAN Storage or individual Solid State Drives (SSDs).

Note: Network-attached storage (NAS) is not supported for these disks.

This topology still has a single point of failure when it comes to supporting users that access the processing console.

Highly available analytics topology

In this topology, both processing console and search nodes are on individual physical servers. Every server component has a back-up and recovery mechanism in the event of an unexpected failure. The following image shows a highly available topology:


Web
Browser
http(s)
HTTP Load Balancer
IBM SmartCloud Analytics Embedded Node 1
Processing Console Node 01
Search Client
IBM SmartCloud Analytics Embedded Node 2
Processing Console Node 02
Search Client
IBM SmartCloud Analytics Embedded Node 3
Search Node 01
Disk
IBM SmartCloud Analytics Embedded Node 4
Search Node 02
Disk

An HTTP load balancer in the front makes sure that more number of concurrent users are supported equally across the two processing console nodes.

The search clients in the processing nodes are part of the search cluster. The routing of search requests to the search nodes is automatically handled by the search cluster.

If one of the search nodes or one of the console nodes fails, IBM SmartCloud Analytics Embedded is still functional for users. This setup is an optimized search cluster setup and is best for production environments.

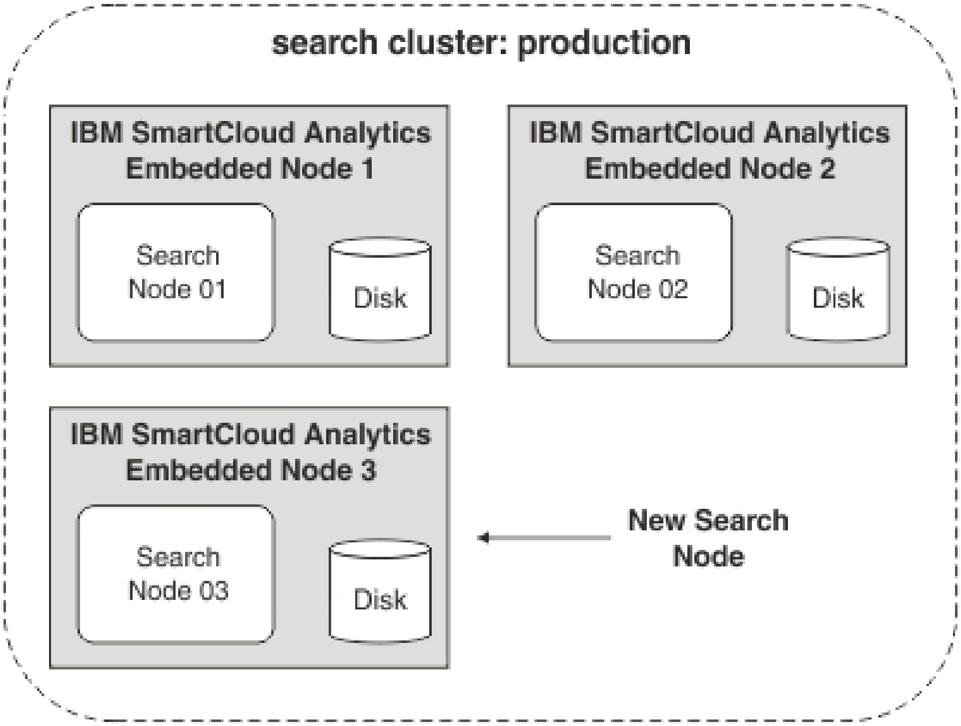
IBM SmartCloud Analytics Embedded scalability:

The individual nodes that participate in processing console or search server are elastically scalable. More nodes can be added to scale up the performance of the analytics component. For enhanced storage and search performance, new search nodes are added. For increased number of concurrent users, processing console nodes can be added.

Adding search nodes for enhanced search performance and storage

A search cluster consists of one or more search nodes and works in primary, secondary modes to give a complete balanced and distributed search cluster.

The primary search node is also referred to as master search node. The first node to come up in a cluster is the primary or master search node. That node is responsible for distributing search and indexing tasks for the participating secondary nodes in the search cluster. The following image shows how to add new search nodes:


search cluster: production
IBM SmartCloud Analytics Embedded Node 1
Search Node 01
Disk
IBM SmartCloud Analytics Embedded Node 2
Search Node 02
Disk
IBM SmartCloud Analytics Embedded Node 3
Search Node 03
Disk
New Search Node

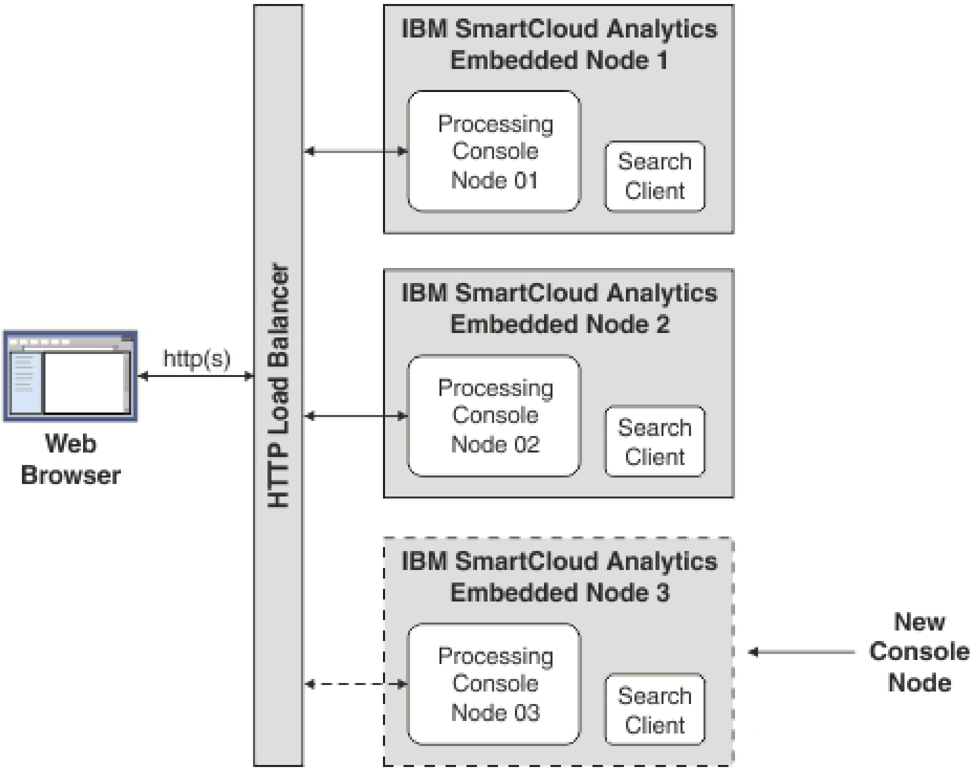
The new search node must be in the same subnet as the primary or master node for it be automatically added to the search cluster. No other manual configurations are necessary.

When a new search node is added, part of the data is transferred to the new node. The disk space that is associated with this node also becomes available for the total space available for storing the Worklight Server and device data.

This type of scaling allows search clusters to be elastically scaled to tune the requirements of growing mobile analytics data. To manage the overall size of the search cluster's storage, see "IBM SmartCloud Analytics Embedded data accumulation management" on page 949.

Adding processing console nodes for increased concurrent users

A console node can communicate seamlessly with the search node or cluster and is stateless in nature. It becomes easier to have multiple console nodes handle incoming user requests through the web browser for viewing the Worklight Analytics dashboard. The following image shows how to add new processing console nodes:


IBM SmartCloud Analytics Embedded Node 1
Processing Console Node 01
Search Client
IBM SmartCloud Analytics Embedded Node 2
Processing Console Node 02
Search Client
IBM SmartCloud Analytics Embedded Node 3
Processing Console Node 03
Search Client
New Console Node
HTTP Load Balancer
http(s)
Web Browser

A new processing console node needs configuration updates to the load balancer in front so it can route HTTP(S) traffic from users to the new HTTP node.

If a search node process goes down, the secondary node comes up as a primary node. The data is recovered on the secondary server from the replicas that are maintained in the secondary node.

IBM SmartCloud Analytics Embedded data accumulation management:

The amount of data that is accumulated by IBM SmartCloud Analytics Embedded depends on a number of factors.

These factors include but are not limited to:
- The log level that is configured on the server
- Whether client applications are calling the `WL.Analytics.log` function and with what frequency
- Whether client applications are calling the `WL.Client.logActivity` function and with what frequency
- How the client application is used (such as how often it is brought into the foreground and how long it remains in the foreground)
- Whether the Worklight Location Services feature is enabled and how it is configured
- The number of clients
- If client applications are failing, and how frequently
- Whether server adapters are calling the `WL.Server.logActivity` function and with what frequency

For example, consider a client application that was created with no additional customizations or features enabled. The usage pattern is that the application is used five times a day, and stays in the foreground for 3 minutes each time it is used. For this scenario, IBM SmartCloud Analytics Embedded accumulates approximately 35 KB of data per day.

If the previous example application additionally makes 10 calls to the `WL.Analytics.log` function each time it enters the foreground, IBM SmartCloud Analytics Embedded accumulates approximately 91 KB of data per day.

The search index is configured for 300 GB by default. This value is configurable by using the `updateTenant.py` script as follows:

```
<INSTALL_LOCATION>/searchengine/python updateTenant.py -t worklight -v <desired size in GB>
```

The search engine fails to respond if the disk is full on the server that is hosting IBM SmartCloud Analytics Embedded. The data in the search index can be pruned by using the `pruneData.py` script. The script cuts down the older data when the search index exceeds the defined volume.

The `pruneData.py` script can be run by administrators periodically by using the following command:

```
python pruneData.py
```

For example, if your current disk usage is around 350 GB, then by running the `pruneData.py` utility, the oldest data is deleted to bring down the disk space to ~275 GB.

The command to run the `pruneData.py` script can also be scheduled to run periodically. Create a shell script that starts the command and schedule the script to run at a specific interval. You can use `crontab` or other similar time-based service feature.

In addition to pruning the data in the search index that is accumulated, it is also important to check the size of log files. Ensure that the console logs and the search engine logs are regularly monitored and cleaned up to avoid failures due to insufficient disk space. The console logs can be found in `<INSTALL_LOCATION>/HTTPServer/logs`. The search engine logs can be found in the location that you chose during installation.

Enabling analytics

To use the Analytics feature since IBM Worklight V6.1.0, you must take steps to enable it.

Before you begin

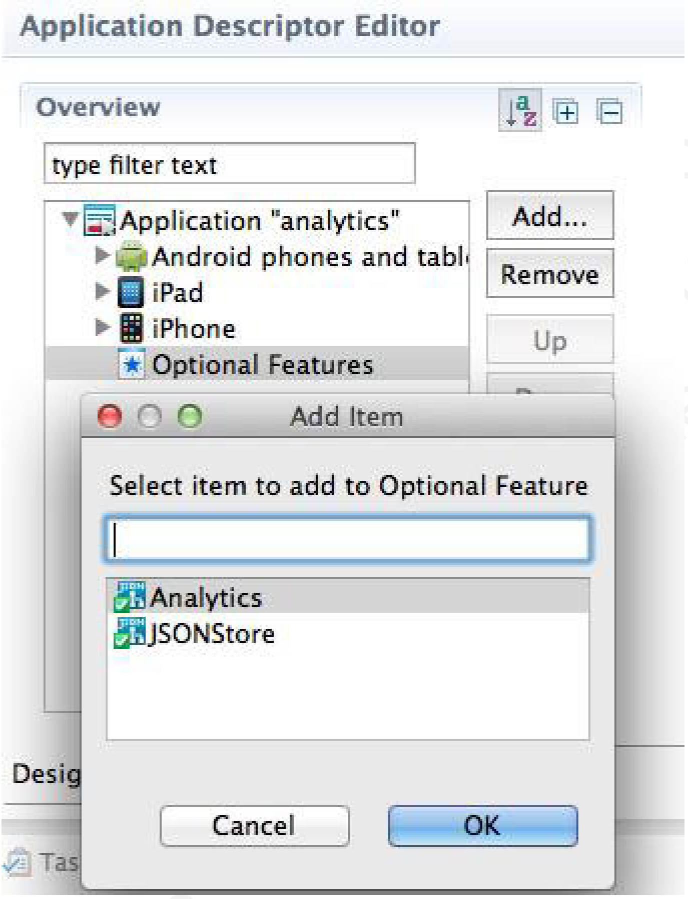
Ensure that you configured your server for analytics as explained in “Configuring Worklight Server for analytics” on page 215.

About this task

The Analytics feature is an optional feature since IBM Worklight V6.1.0. To use the Analytics feature, 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 `Analytics`.
7. Click **Ok**.
8. Under **Worklight Project**, right-click the folder titled with your application name.
9. Select **Run As...**.
10. Click **Run on Worklight Development Server**.



Select item to add to Optional Feature

Figure 178. Enabling analytics

Results

You have modified the `application-descriptor.xml` file to enable the Analytics feature.

What to do next

For more information on sending client application analytic data to the IBMWorklight Server, see “Enabling analytic data for new applications.” If you need to enable analytics for existing applications, see “Enabling analytic data for existing applications.”

Related concepts:

“Troubleshooting analytics” on page 966
Find solutions to problems with IBM Worklight analytics features.

Enabling analytic data for new applications

You must modify your application to send client application analytic data to the IBM Worklight Server.

Note: Ensure that you enabled the Analytics feature as explained in “Enabling analytics” on page 950.

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 that the default value of `analytics.enabled` is false. To enable sending captured application crash data, and analytic data that is recorded by the `WL.Analytics.log` function call, you must either set the option to true or call the `WL.Analytics.enable` function.

In IBM Worklight, client application crashes are recorded and analytic data are sent to the IBM Worklight Server when the client analytic feature is enabled. If the IBM Worklight Server is properly configured, this application crash data is then forwarded to IBM SmartCloud Analytics Embedded. 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 the IBM Worklight Server.

Note: No data is sent to the IBM Worklight Server until the application is connected to the server. You can set `connectOnStartup : true` in the `wlInitOptions` object in the `[appFolder]/common/js/initOptions.js` file, or successfully call an adapter in the IBM Worklight Server.

The full `WL.Analytics` API is described in WL.Analytics.

Enabling analytic data for existing applications

You must modify applications that were created in earlier versions of IBM Worklight to send client application analytic data to the IBM Worklight Server.

Note: Ensure that you enabled the Analytics feature as explained in “Enabling analytics” on page 950.

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 the [appFolder]/common/js/initOptions.js file:

```
analytics : {
  enabled: true
  //url : //
}
```

Uncomment the url property and provide your server URL information if you send data to a server other than the Worklight Server.

You must update the JNDI configuration for your WAR file to set the other required properties, as explained in "Configuring Worklight Server for analytics" on page 215.

Note: Applications that use the IBM Worklight V5.0.6 or earlier that contained Tealeaf client libraries must be manually upgraded into a new IBM Worklight V6.0.0 project. For more information about how to migrate applications, see Migration of projects using Tealeaf libraries.

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 tab in the Worklight Console

The Analytics tab in the Worklight Console is the user interface from which you can search for analytics data and see results from your IBM Worklight system.

The Analytics tab 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

To read a description for each chart, click the icon in the Dashboard view next to the name of the chart.

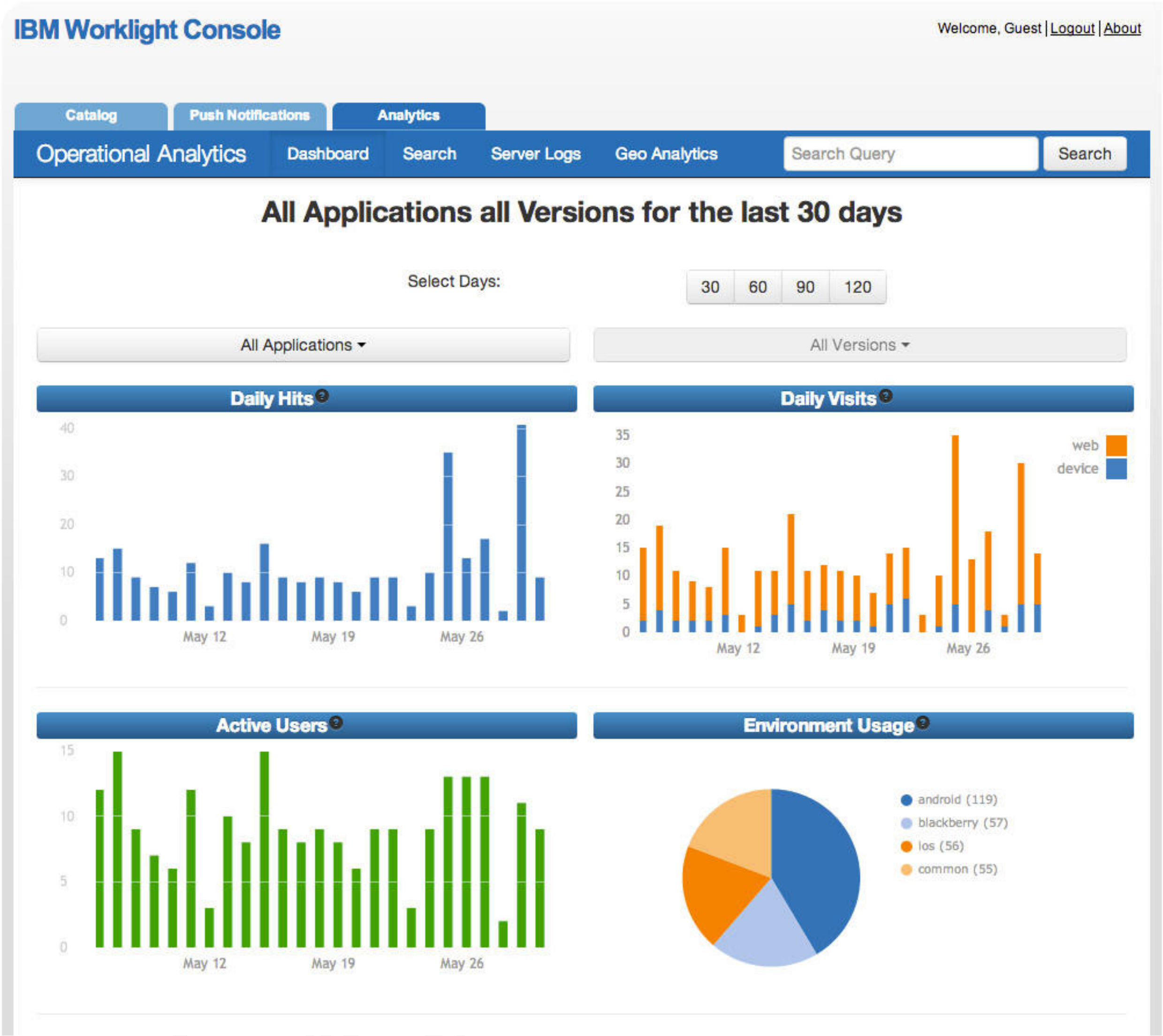


Figure 179. The Dashboard view is in the Analytics tab of the IBM Worklight Console

Search view:

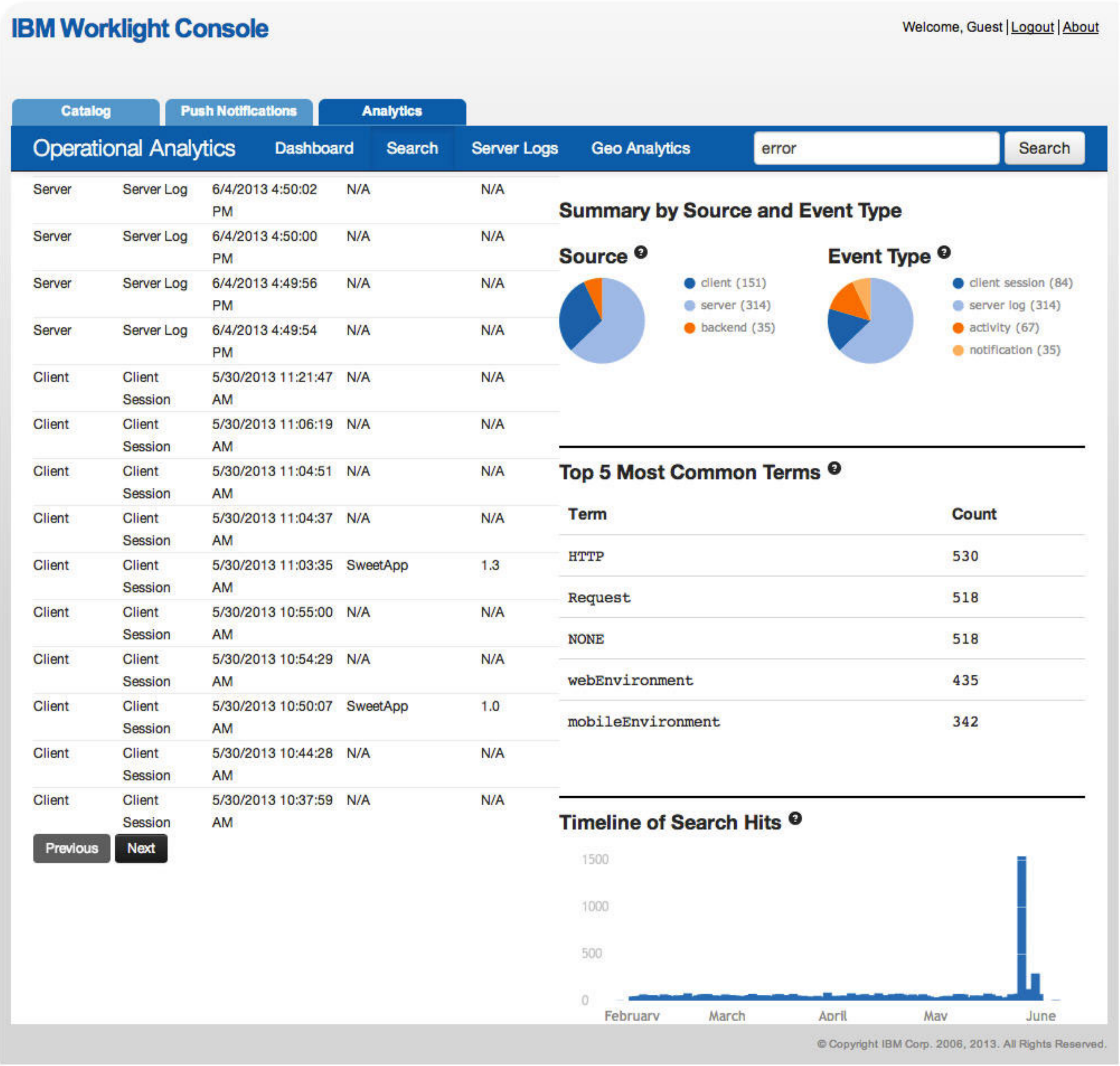
The Search view in the Analytics page enables you to search for logged events from Worklight server.

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.

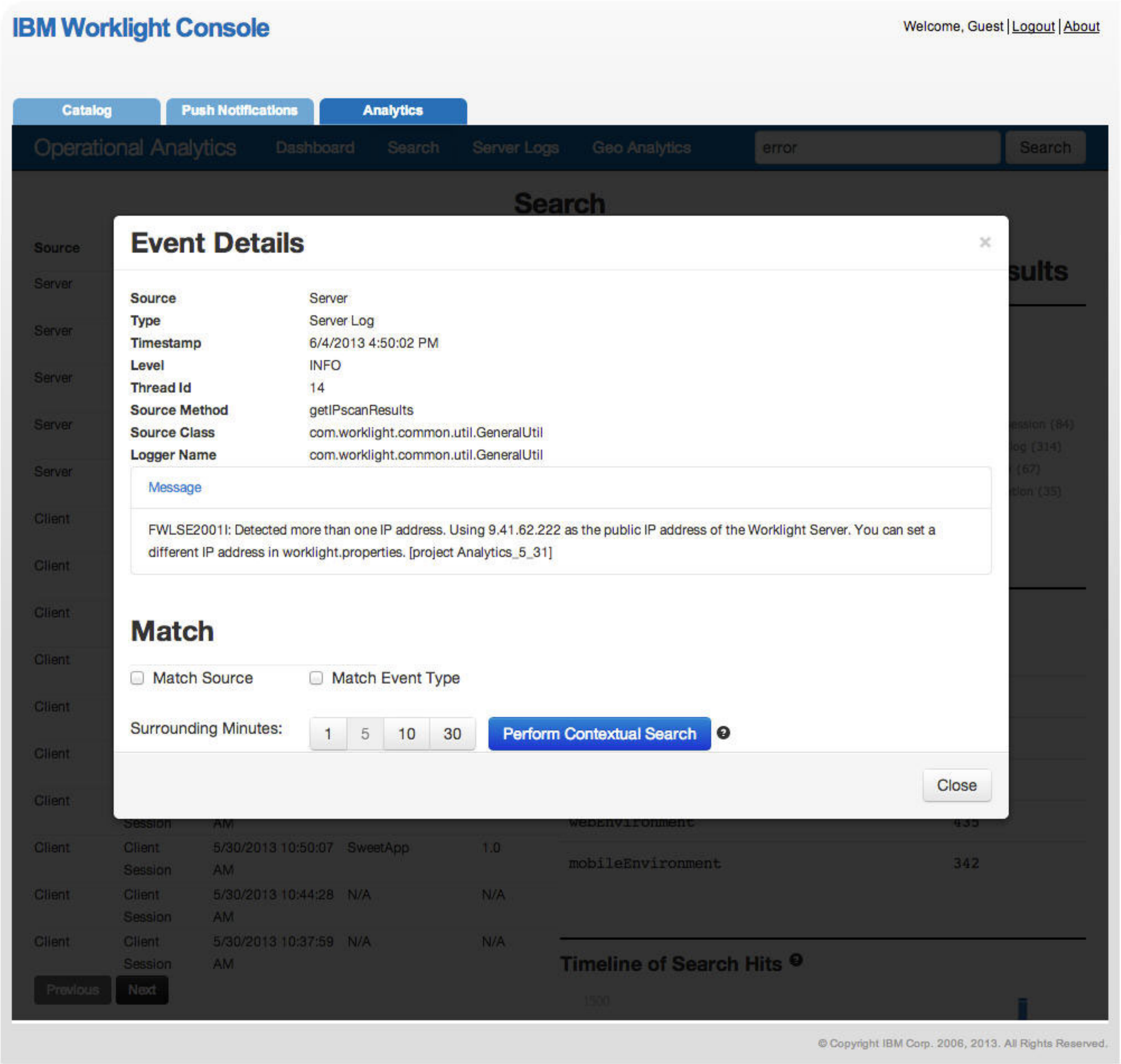


Server	Server Log	6/4/2013 4:50:02 PM	N/A	N/A
Server	Server Log	6/4/2013 4:50:00 PM	N/A	N/A
Server	Server Log	6/4/2013 4:49:56 PM	N/A	N/A
Server	Server Log	6/4/2013 4:49:54 PM	N/A	N/A
Client	Client Session	5/30/2013 11:21:47 AM	N/A	N/A
Client	Client Session	5/30/2013 11:06:19 AM	N/A	N/A
Client	Client Session	5/30/2013 11:04:51 AM	N/A	N/A
Client	Client Session	5/30/2013 11:04:37 AM	N/A	N/A
Client	Client Session	5/30/2013 11:03:35 AM	SweetApp	1.3
Client	Client Session	5/30/2013 10:55:00 AM	N/A	N/A
Client	Client Session	5/30/2013 10:54:29 AM	N/A	N/A
Client	Client Session	5/30/2013 10:50:07 AM	SweetApp	1.0
Client	Client Session	5/30/2013 10:44:28 AM	N/A	N/A
Client	Client Session	5/30/2013 10:37:59 AM	N/A	N/A

Term	Count
HTTP	530
Request	518
NONE	518
webEnvironment	435
mobileEnvironment	342

Figure 180. Search results

When you click a search result, the page displays detailed information about the individual hit.



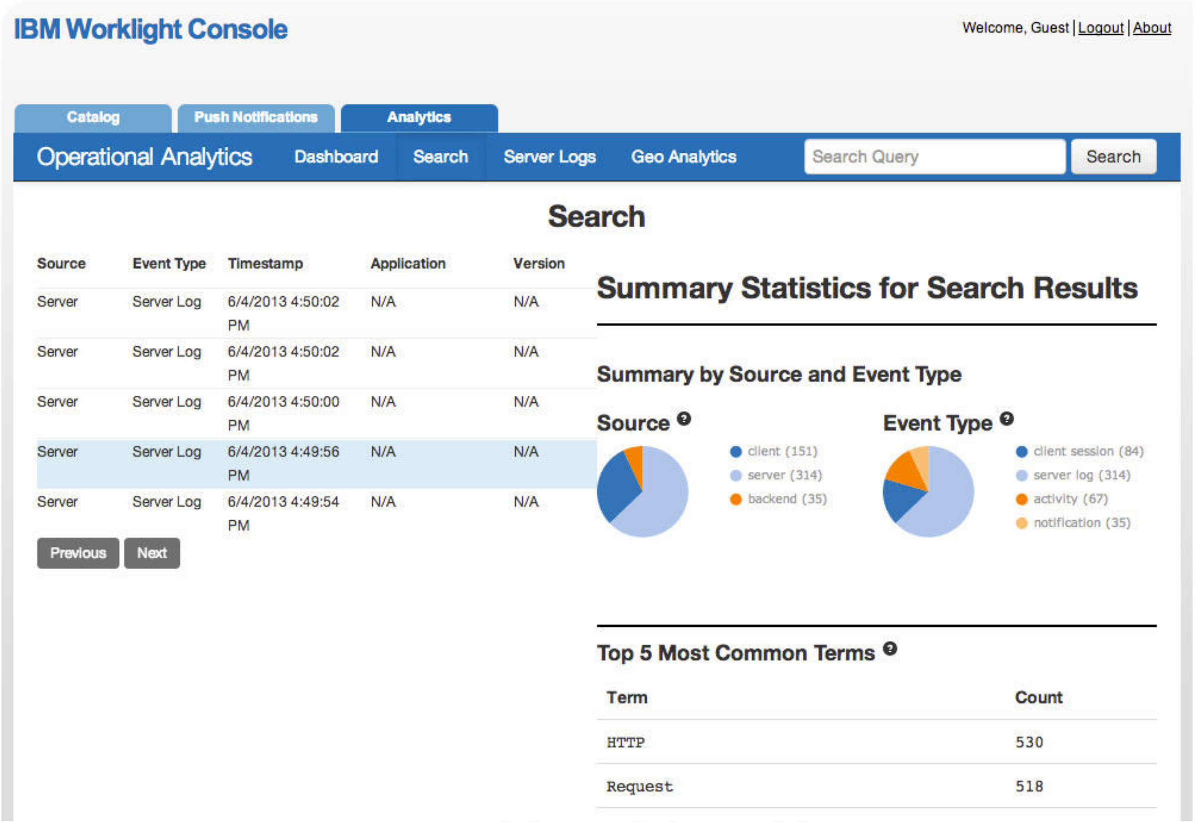
Source	Server
Type	Server Log
Timestamp	6/4/2013 4:50:02 PM
Level	INFO
Thread Id	14
Source Method	getIPscanResults
Source Class	com.worklight.common.util.GeneralUtil
Logger Name	com.worklight.common.util.GeneralUtil

Message

FWLSE2001I: Detected more than one IP address. Using 9.41.62.222 as the public IP address of the Worklight Server. You can set a different IP address in worklight.properties. [project Analytics_5_31]

Figure 181. 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.



Search

Source	Event Type	Timestamp	Application	Version
Server	Server Log	6/4/2013 4:50:02 PM	N/A	N/A
Server	Server Log	6/4/2013 4:50:02 PM	N/A	N/A
Server	Server Log	6/4/2013 4:50:00 PM	N/A	N/A
Server	Server Log	6/4/2013 4:49:56 PM	N/A	N/A
Server	Server Log	6/4/2013 4:49:54 PM	N/A	N/A

Summary Statistics for Search Results

Summary by Source and Event Type

Top 5 Most Common Terms

Term	Count
HTTP	530
Request	518

Figure 182. 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 IBM SmartCloud Analytics Embedded. 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:

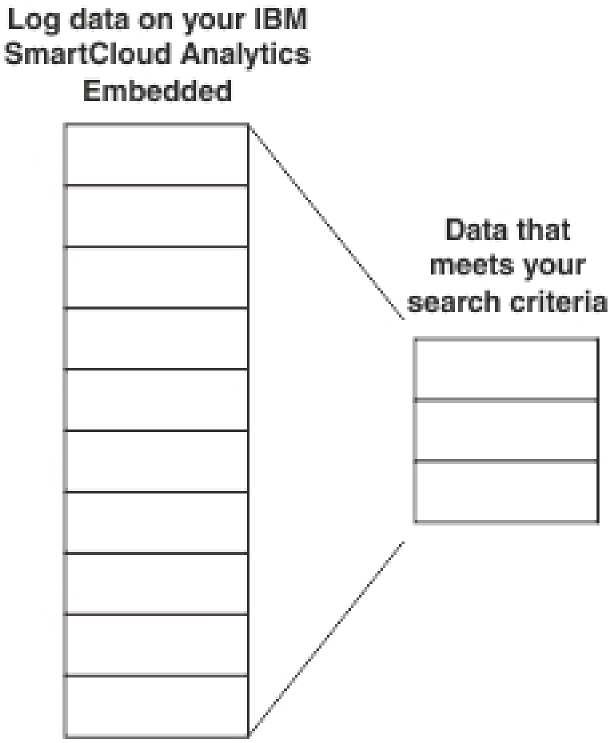

Log data on your IBM SmartCloud Analytics Embedded
Data that meets your search criteria

Figure 183. 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.

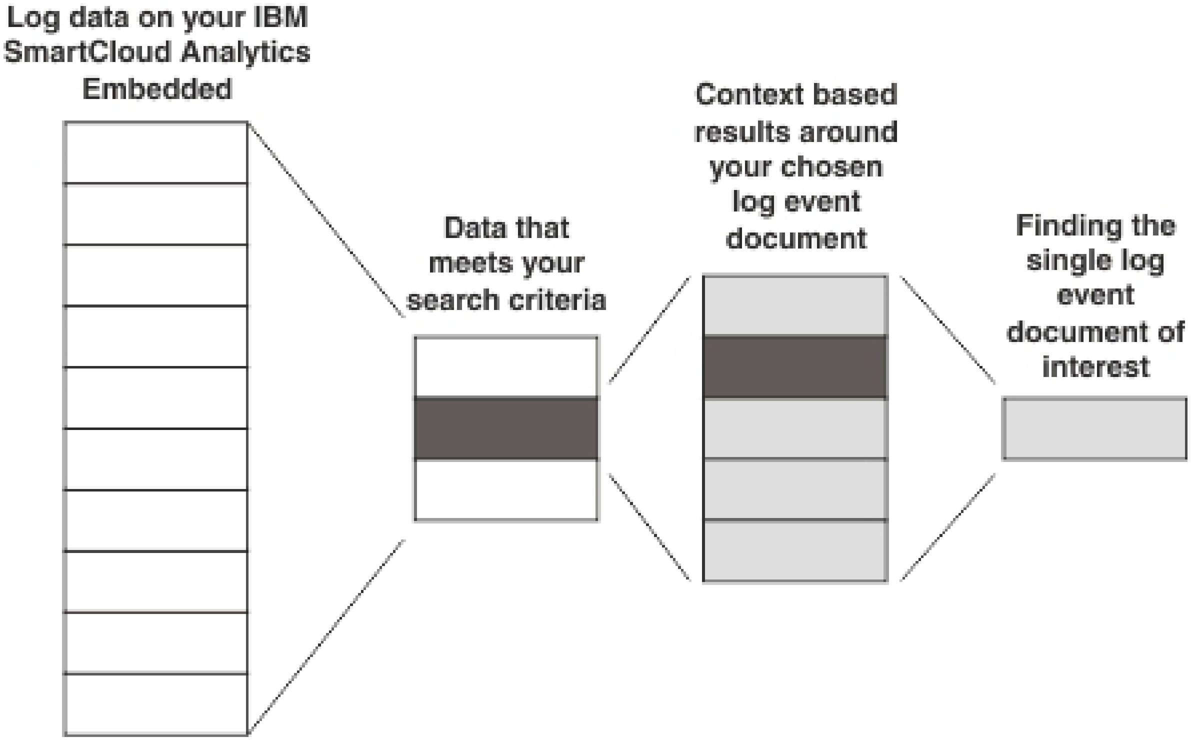
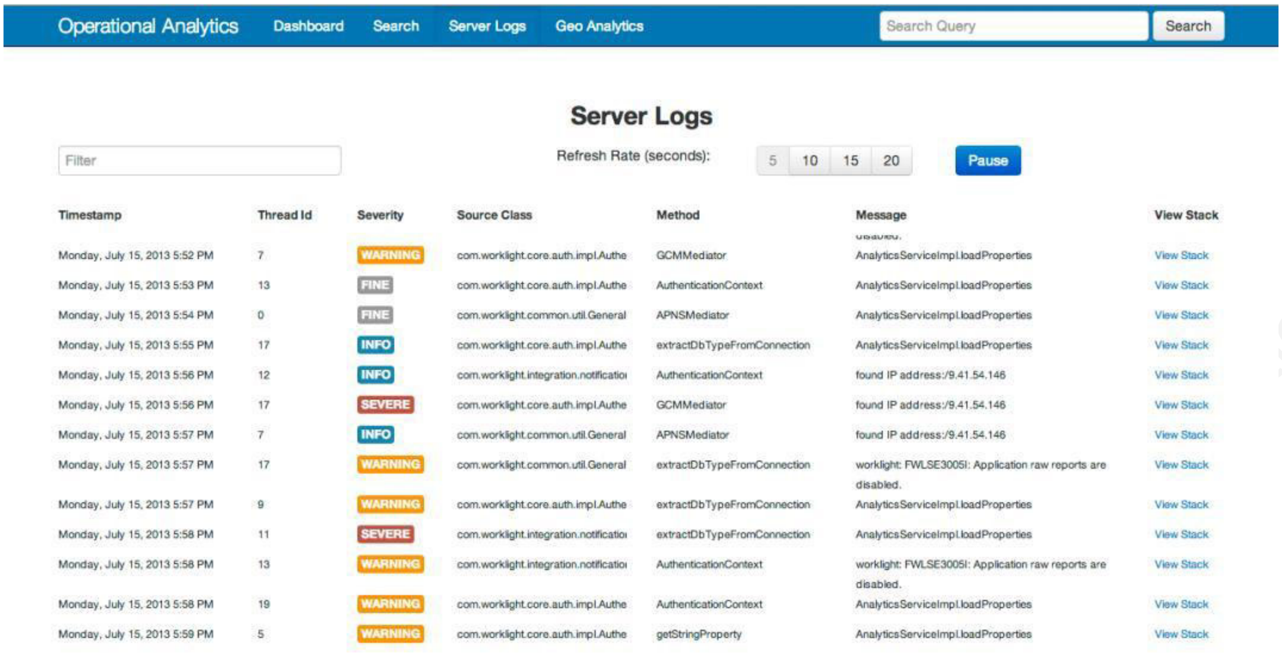

Log data on your IBM SmartCloud Analytics Embedded
Data that meets your search criteria
Context based results around your chosen log event document
Finding the single log event document of interest

Figure 184. 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

Timestamp	Thread Id	Severity	Source Class	Method	Message	View Stack
					disabled.	
Monday, July 15, 2013 5:52 PM	7	WARNING	com.worklight.core.auth.impl.Authe	GCMMediator	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:53 PM	13	FINE	com.worklight.core.auth.impl.Authe	AuthenticationContext	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:54 PM	0	FINE	com.worklight.common.util.General	APNSMediator	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:55 PM	17	INFO	com.worklight.core.auth.impl.Authe	extractDbTypeFromConnection	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:56 PM	12	INFO	com.worklight.integration.notificatio	AuthenticationContext	found IP address:/9.41.54.146	View Stack
Monday, July 15, 2013 5:56 PM	17	SEVERE	com.worklight.core.auth.impl.Authe	GCMMediator	found IP address:/9.41.54.146	View Stack
Monday, July 15, 2013 5:57 PM	7	INFO	com.worklight.common.util.General	APNSMediator	found IP address:/9.41.54.146	View Stack
Monday, July 15, 2013 5:57 PM	17	WARNING	com.worklight.common.util.General	extractDbTypeFromConnection	worklight: FWLSE3005I: Application raw reports are disabled.	View Stack
Monday, July 15, 2013 5:57 PM	9	WARNING	com.worklight.core.auth.impl.Authe	extractDbTypeFromConnection	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:58 PM	11	SEVERE	com.worklight.integration.notificatio	extractDbTypeFromConnection	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:58 PM	13	WARNING	com.worklight.integration.notificatio	AuthenticationContext	worklight: FWLSE3005I: Application raw reports are disabled.	View Stack
Monday, July 15, 2013 5:58 PM	19	WARNING	com.worklight.core.auth.impl.Authe	AuthenticationContext	AnalyticsServiceImpl.loadProperties	View Stack
Monday, July 15, 2013 5:59 PM	5	WARNING	com.worklight.core.auth.impl.Authe	getStringProperty	AnalyticsServiceImpl.loadProperties	View Stack

Figure 185. 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 API” on page 696 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

Set up the required permissions, as indicated in “Location services permissions” on page 661, 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.

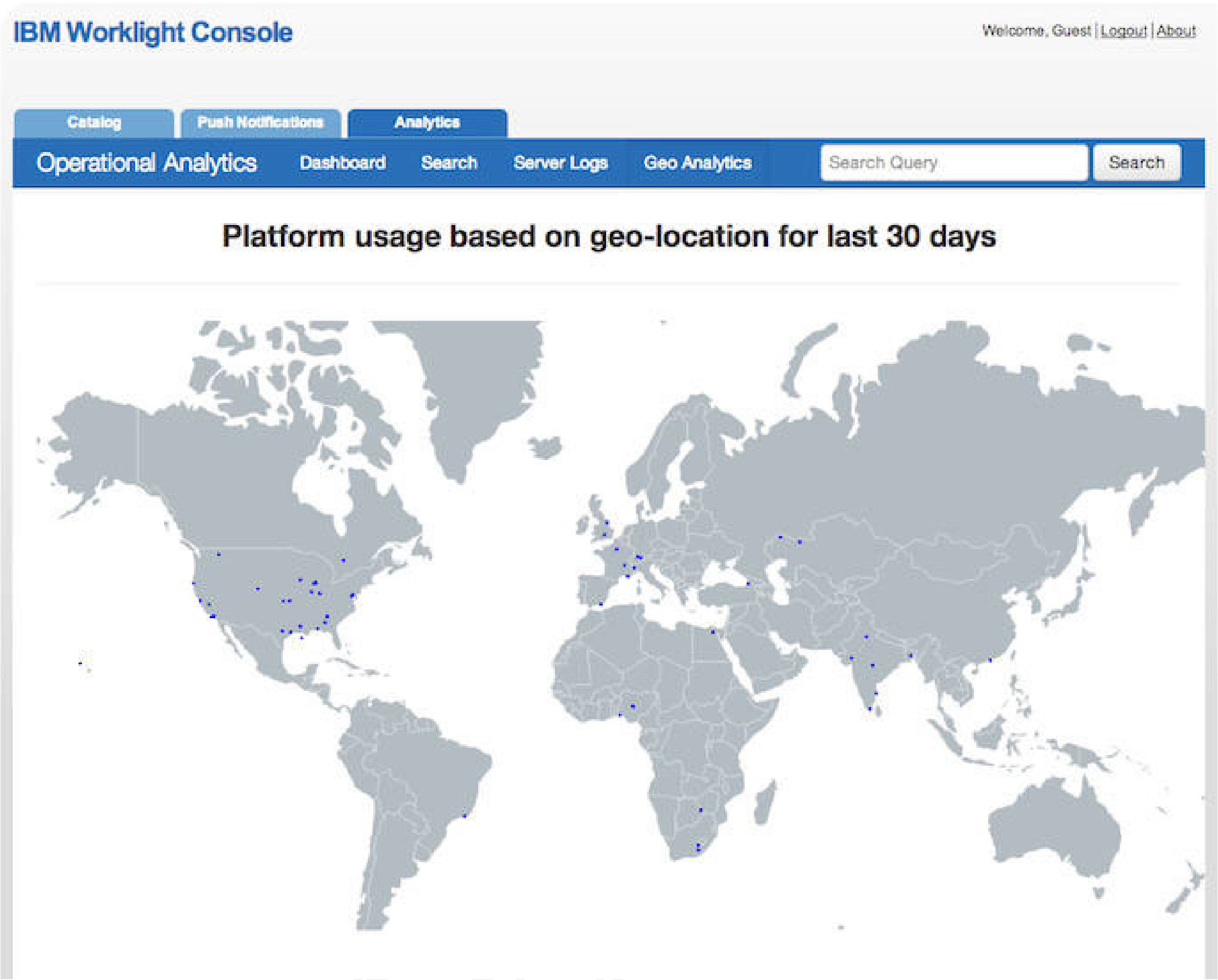


Figure 186. Geospatial information is displayed for clients that are configured to include it

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 IBM SmartCloud Analytics Embedded. 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 SmartCloud Analytics Embedded by using `HTTP POST`. If this connection experiences socket timeouts or other failure, the data is discarded.

In a production environment, the Worklight Server queues analytics data before it sends the data to IBM SmartCloud Analytics Embedded. This behavior means that there might be a delay before the analytics data that is sent by the Worklight Client is visible on IBM SmartCloud Analytics Embedded. In the Worklight Studio development environment when you use the Worklight Development Server, the analytics data is not queued by default. Because of this difference, the same type of delay before the analytics data is visible on IBM SmartCloud Analytics Embedded no longer exists.

When the server is unusually busy, Worklight Server buffers data that is designated for IBM SmartCloud Analytics Embedded. You can tune IBM Worklight to adjust this data loss, taking advantage of the memory capacity of Worklight Sever and speed of the network connection from Worklight Server to IBM SmartCloud Analytics Embedded. For more information see “Worklight Server throughput tuning” on page 964.

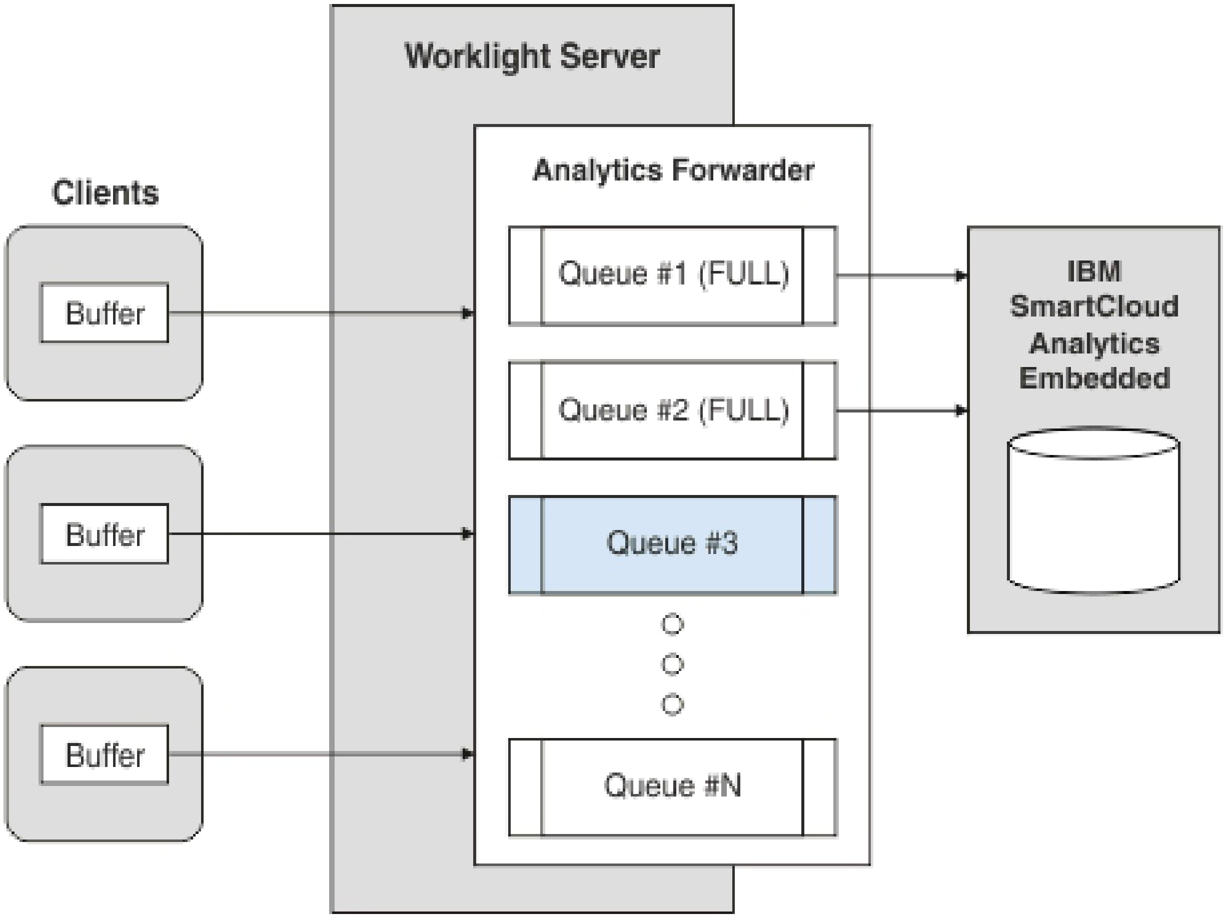


Figure 187. Architectural overview of analytic data throughput.

Managing data throughput and accumulation

You can make configuration changes to adjust how data accumulates in IBM SmartCloud Analytics Embedded (operational analytics) server.

The amount of data that is accumulated by the operational analytics server 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 operational analytics server 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 operational analytics server 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 operational analytics server 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 IBM SmartCloud Analytics Embedded.

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 the operational analytics server:

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 SmartCloud Analytics Embedded (operational analytics) server 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 operational analytics server.

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 `logActivity` function call, as defined in the WL.Client class, to record any activity that you specify, and send the data to the IBM WebSphere Analytics Platform (operational analytics) server (if configured), and the raw reports database (if configured).

The `log` method, as defined in the WL.Analytics class, is a richer way to get data into the operational analytics server than the `logActivity` function call, as defined in the WL.Client class. Using the `log` method, as defined in the WL.Analytics class, 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 operational analytics server (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 Analytics tab in the Worklight Console.

The `log` method, as defined in the WL.Analytics class, 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. The `logActivity` function call, as defined in the WL.Client class, sends instantly when called. If you want to record large amounts of custom analytics data, use the `log` method, as defined in the WL.Analytics class.

Table 185. Activity log settings

	`WL.Analytics.log`	**`WL.Client.logActivity`**
Destination	operational analytics server only	operational analytics server 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 operational analytics server, 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 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.

Troubleshooting analytics

Find solutions to problems with IBM Worklight analytics features.

Table 186. Analytics troubleshooting guidelines

Problem	**Actions to take**
No analytics content appears in the **Analytics** tab on the IBM Worklight Console	Ensure that the `wl.analytics.url` 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 Analytics tab is not present in the IBM Worklight Console.	Ensure that the `wl.analytics.url` 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 WL.Logger.
The server logs tab shows no server logs.	Server logs are not sent if the `wl.analytics.logs.foward` property is set to false in your JNDI configuration for this war. The default value for this property is `true`.

Table 186. Analytics troubleshooting guidelines (continued)

Problem	Actions to take
Analytics content is slow to appear in the **Analytics** 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 964 and WL.Analytics.
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 964.

For more information, see “Troubleshooting Worklight Server” on page 216

Reports database

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 939

Note: Enabling the BIRT feature is redundant if you already use the IBM WebSphere Analytics Platform.

IBM Worklight provides three 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 968.

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 972.

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 974.

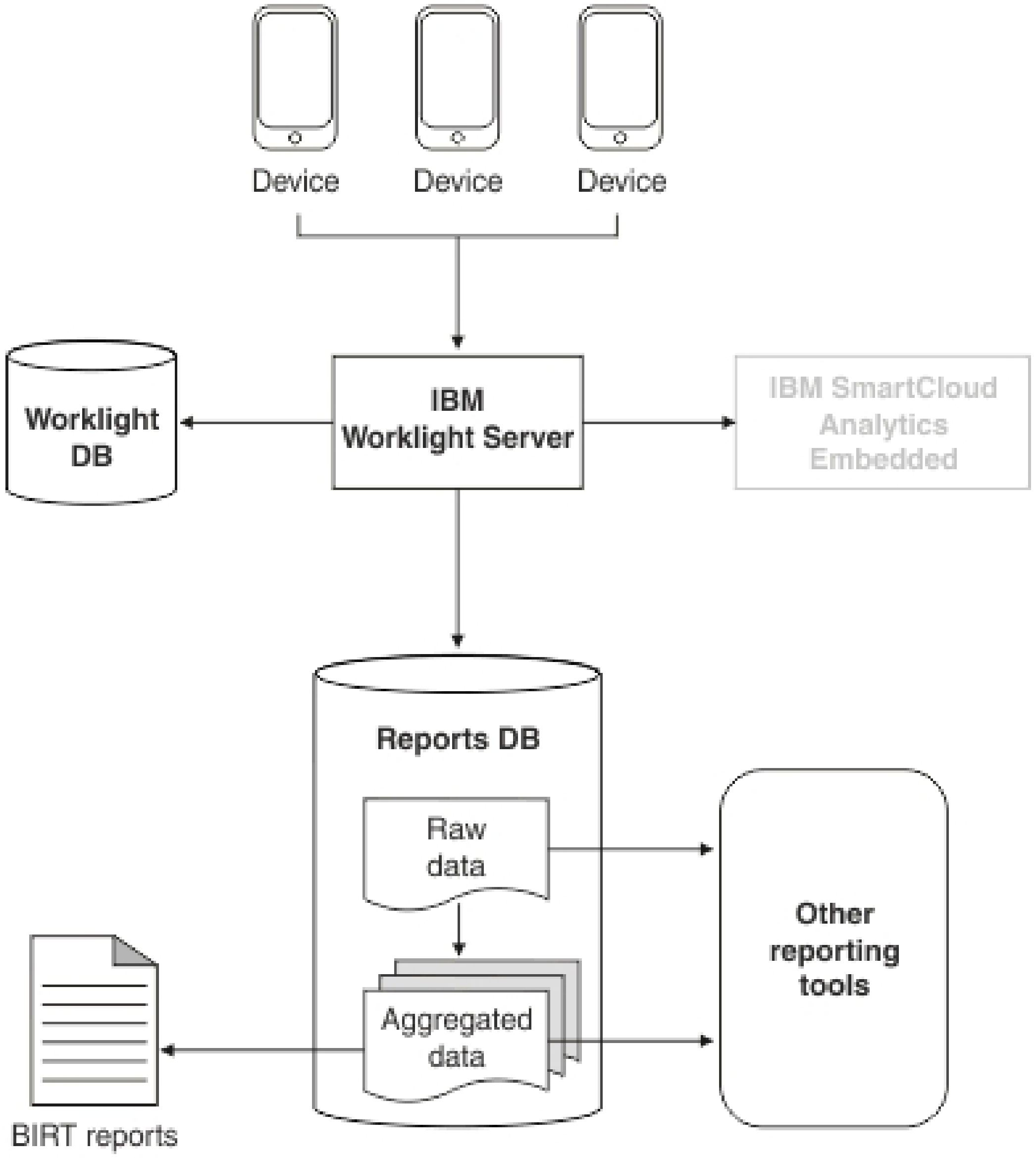


Figure 188. High-level overview of the reports architecture

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.

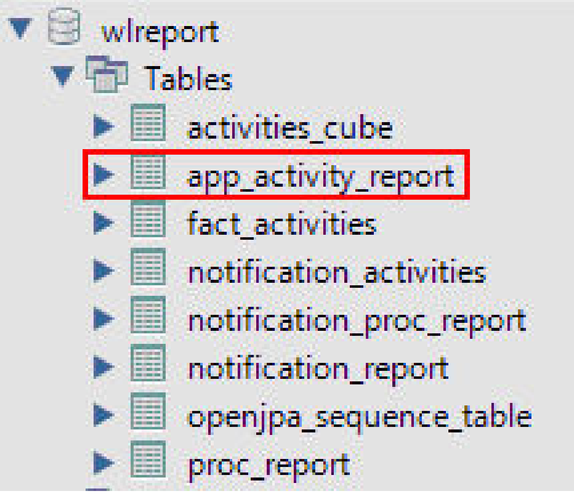
About this task

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 774. For information about creating databases and schemas in a production environment, see "Creating and configuring the databases" on page 719.
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 example.
      ```
      ##################################################
      # 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 784.
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.



wlreport
Tables
activities_cube
app_activity_report
fact_activities
notification_activities
notification_proc_report
notification_report
openjpa_sequence_table
proc_report

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
GADGET_VERSION	Application version
ACTIVITY	Activity type
ENVIRONMENT	Application environment name (iPhone, Android, and so on)
SOURCE	User identifier
ADAPTER	IBM Worklight adapter name
PROC	IBM Worklight adapter procedure name
USERAGENT	User agent from HTTP header of client device
SESSION_ID	A unique identifier for the user's session on the server
IP_ADDRESS	IP address of the client
DEVICE_ID	A unique device ID
DEVICE_MODEL	Manufacturer model, for example Galaxy I9000
DEVICE_OS	Device operating system version
LONGITUDE	The longitude of the device. Requires that ongoing acquisition is enabled for Geo.
LATITUDE	The latitude of the device. Requires that ongoing acquisition is enabled for Geo.
POS_USER_TIME	The local time on the device when the latest position information (longitude and latitude) were updated. Requires that ongoing acquisition is enabled for Geo.
WIFI_APS	The access points visible on the device. Requires that ongoing acquisition is enabled for WiFi.

Column	Description
WIFI_CONNECTED_SSID	The SSID (network identification) of the connected WiFi access point. Requires that ongoing acquisition is enabled for WiFi.
WIFI_CONNECTED_MAC	The MAC address of the connected WiFi access point. Requires that ongoing acquisition is enabled for WiFi.
WIFI_USER_TIME	The local time on the device when the latest WiFi information was updated. Requires that ongoing acquisition is enabled for WiFi.
APP_CONTEXT	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
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, 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):

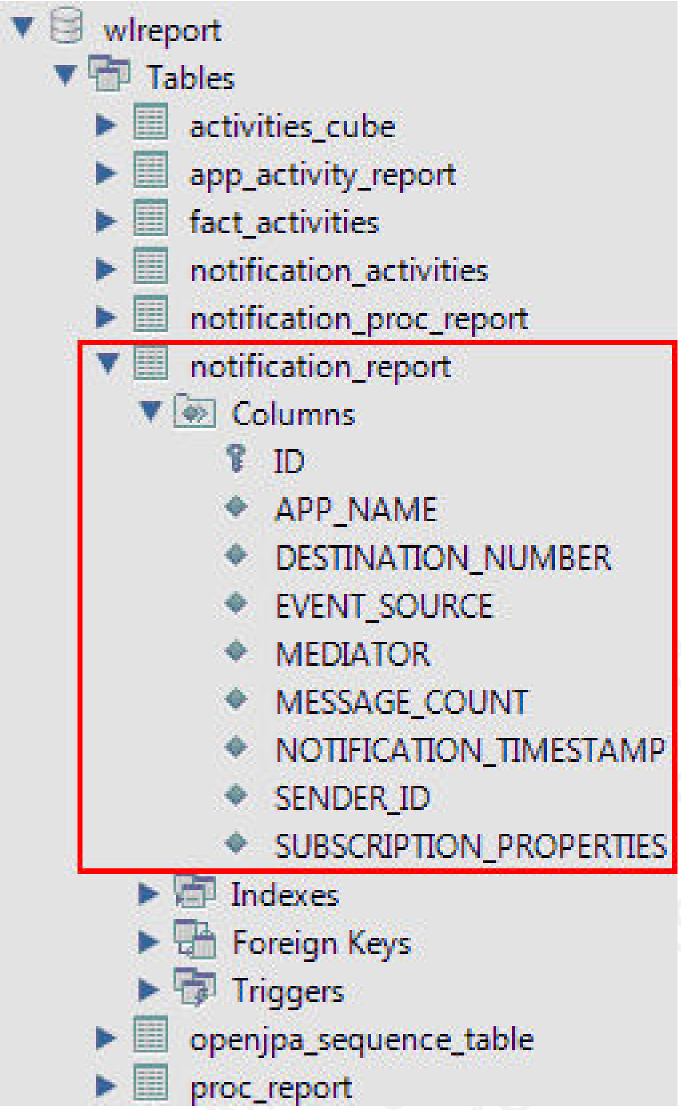
```
DELETE FROM ACTIVITIES_CUBE WHERE ACTIVITY_DATE <= TRUNC(SYSDATE) - 30
```

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 104.

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.



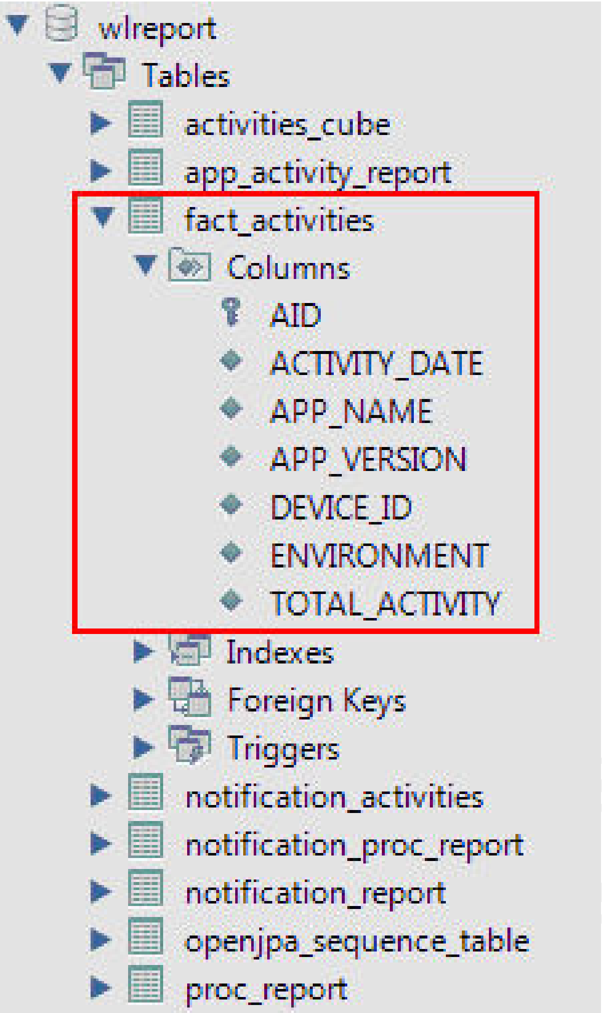
- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
      - Columns
        - ID
        - APP_NAME
        - DESTINATION_NUMBER
        - EVENT_SOURCE
        - MEDIATOR
        - MESSAGE_COUNT
        - NOTIFICATION_TIMESTAMP
        - SENDER_ID
        - SUBSCRIPTION_PROPERTIES
      - Indexes
      - Foreign Keys
      - Triggers
    - openjpa_sequence_table
    - proc_report

Device usage reports

For simpler and faster access to the reports data, Worklight Server runs an analytics data processor task at a default time interval of every 24 hours.

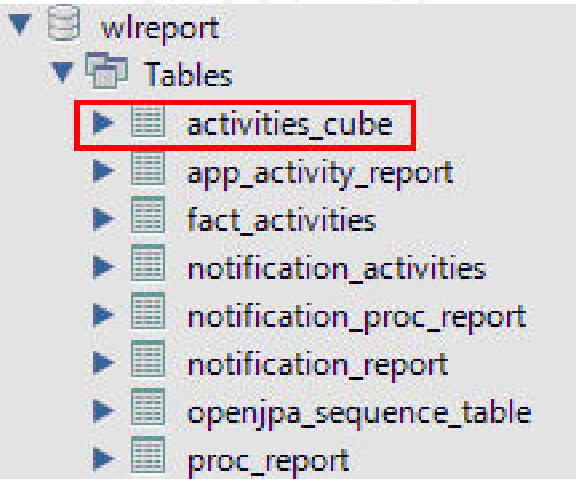
The analytics data processor task retrieves raw entries for the specified time interval from the `app_activity_report` table and processes them to populate the `fact_activities` table.

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.



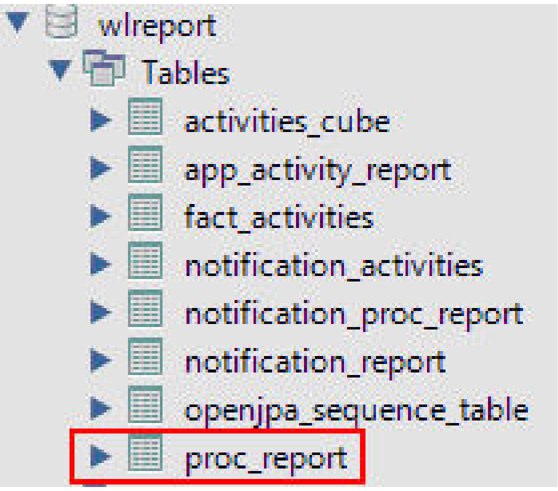
wlreport
Tables
activities_cube
app_activity_report
fact_activities
Columns
AID
ACTIVITY_DATE
APP_NAME
APP_VERSION
DEVICE_ID
ENVIRONMENT
TOTAL_ACTIVITY
Indexes
Foreign Keys
Triggers
notification_activities
notification_proc_report
notification_report
openjpa_sequence_table
proc_report

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.



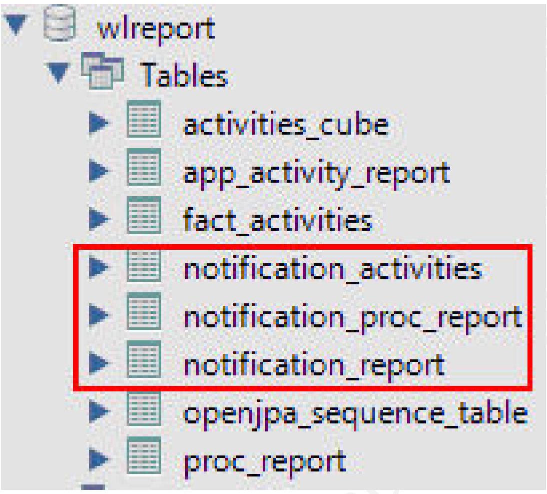
wlreport
Tables
activities_cube
app_activity_report
fact_activities
notification_activities
notification_proc_report
notification_report
openjpa_sequence_table
proc_report

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).



- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
    - openjpa_sequence_table
    - proc_report

In addition, `notification_report` table data is also processed to populate the `notification_activities` table with consolidated data. 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.



- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
    - notification_activities
    - notification_proc_report
    - notification_report
    - openjpa_sequence_table
    - proc_report

The processing interval can be modified by adding the following property to your `worklight.properties` file and setting the required interval in seconds.

```
# Default interval value for analytics processing task
wl.db.factProcessingInterval=86400
```

The processing interval can also be disabled by setting this property to a negative value.

```
# Set to a negative value to disable the analytics processing task
wl.db.factProcessingInterval=-1
```

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 972 and “Using raw data reports” on page 968. 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.

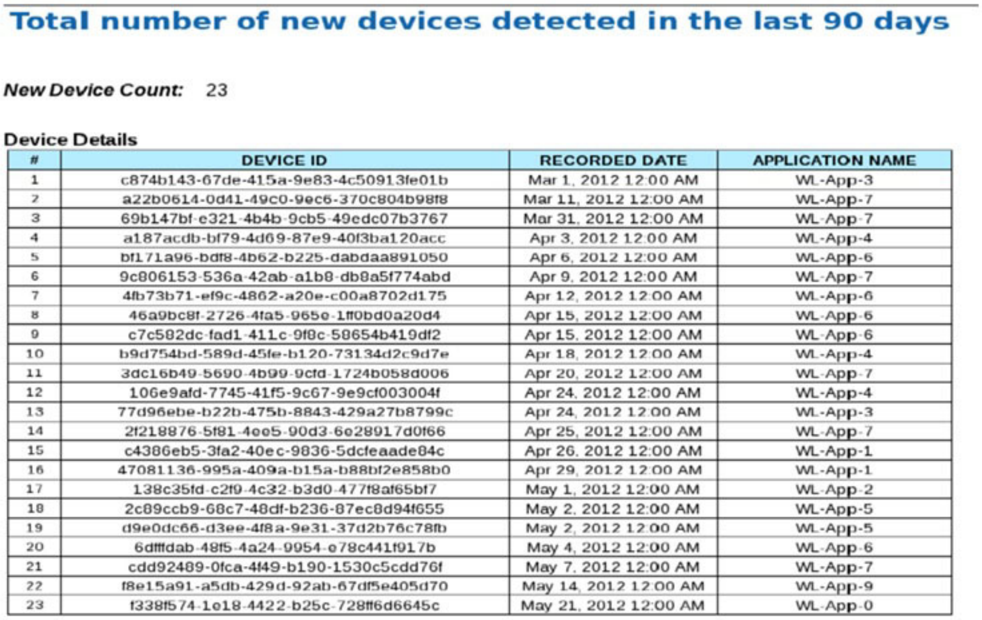
IBM Worklight Customer and Enterprise Editions include the following predefined BIRT reports:

Table 187. 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

Table 187. Predefined BIRT reports (continued)

Report Name	Description	Report file name
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`
License Total New Device Count	A record of unique devices that were connected over a specified period (90 days as default), for licensing purposes.	`report_license_total_device_count.rptdesign`



Total number of new devices detected in the last 90 days

New Device Count: 23

Device Details

#	DEVICE ID	RECORDED DATE	APPLICATION NAME
1	c874b143-67de-415a-9e83-4c50913fe01b	Mar 1, 2012 12:00 AM	WL-App-3
2	a22b0614-0d41-49c0-9ec6-370c804b98f8	Mar 11, 2012 12:00 AM	WL-App-7
3	69b147bf-e321-4b4b-9cb5-49edc07b3767	Mar 31, 2012 12:00 AM	WL-App-7
4	a187acdb-bf79-4d69-87e9-40f3ba120acc	Apr 3, 2012 12:00 AM	WL-App-4
5	bf171a96-bdf8-4b62-b225-dabdaa891050	Apr 6, 2012 12:00 AM	WL-App-6
6	9c806153-536a-42ab-a1b8-db8a5f774abd	Apr 9, 2012 12:00 AM	WL-App-7
7	4fb73b71-ef9c-4862-a20e-c00a8702d175	Apr 12, 2012 12:00 AM	WL-App-6
8	46a9bc8f-2726-4fa5-965e-1ff0bd0a20d4	Apr 15, 2012 12:00 AM	WL-App-6
9	c7c582dc-fad1-411c-9f8c-58654b419df2	Apr 15, 2012 12:00 AM	WL-App-6
10	b9d754bd-589d-45fe-b120-73134d2c9d7e	Apr 18, 2012 12:00 AM	WL-App-4
11	3dc16b49-5690-4b99-9cfd-1724b058d006	Apr 20, 2012 12:00 AM	WL-App-7
12	106e9afd-7745-41f5-9c67-9e9cf003004f	Apr 24, 2012 12:00 AM	WL-App-4
13	77d96ebe-b22b-475b-8843-429a27b8799c	Apr 24, 2012 12:00 AM	WL-App-3
14	2f218876-5f81-4ee5-90d3-6e28917d0f66	Apr 25, 2012 12:00 AM	WL-App-7
15	c4386eb5-3fa2-40ec-9836-5dcfeaade84c	Apr 26, 2012 12:00 AM	WL-App-1
16	47081136-995a-409a-b15a-b88bf2e858b0	Apr 29, 2012 12:00 AM	WL-App-1
17	138c35fd-c2f9-4c32-b3d0-477f8af65bf7	May 1, 2012 12:00 AM	WL-App-2
18	2c89ccb9-68c7-48df-b236-87ec8d94f655	May 2, 2012 12:00 AM	WL-App-5
19	d9e0dc66-d3ee-4f8a-9e31-37d2b76c78fb	May 2, 2012 12:00 AM	WL-App-5
20	6dfffdab-48f5-4a24-9954-e78c441f917b	May 4, 2012 12:00 AM	WL-App-6
21	cdd92489-0fca-4f49-b190-1530c5cdd76f	May 7, 2012 12:00 AM	WL-App-7
22	f8e15a91-a5db-429d-92ab-67df5e405d70	May 14, 2012 12:00 AM	WL-App-9
23	f338f574-1e18-4422-b25c-728ff6d6645c	May 21, 2012 12:00 AM	WL-App-0

Figure 189. 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 options.

- The Eclipse report designer plug-in. For instructions, see "BIRT in Eclipse" on page 983
- 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.

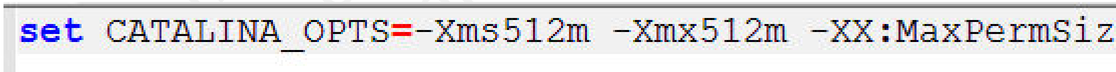
About this task

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.

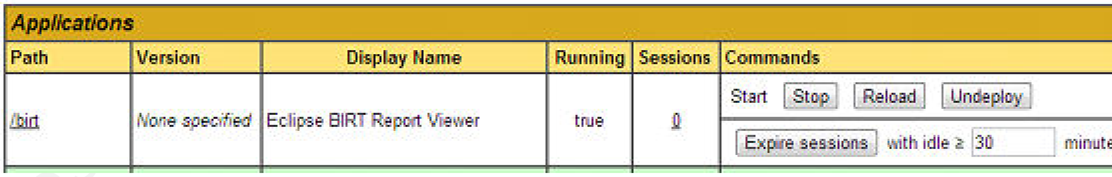
Procedure

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 isan 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.



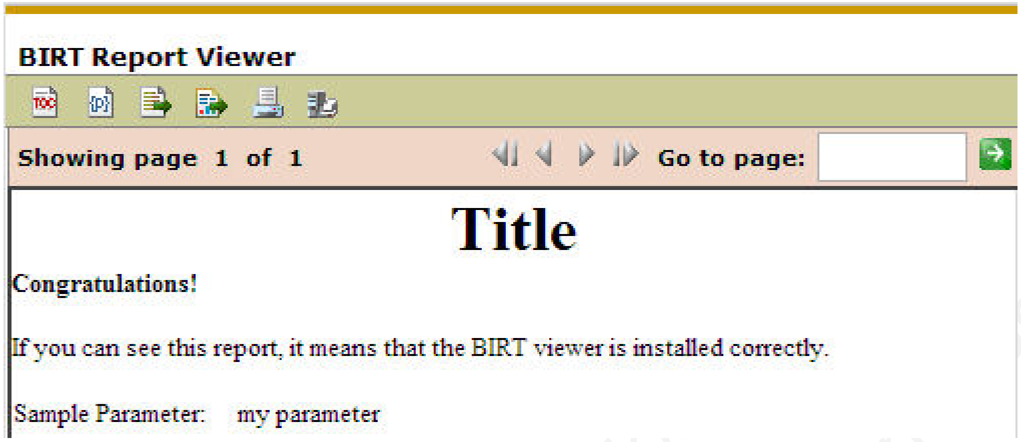
```
set CATALINA_OPTS=-Xms512m -Xmx512m -XX:MaxPermSiz
```

8. Restart your Tomcat.
9. Go to theTomcat manager application at `http://your-server/manager/` to verify that the BIRT Reports application started.



Applications					
Path	Version	Display Name	Running	Sessions	Commands
/birt	None specified	Eclipse BIRT Report Viewer	true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minute

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`.



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

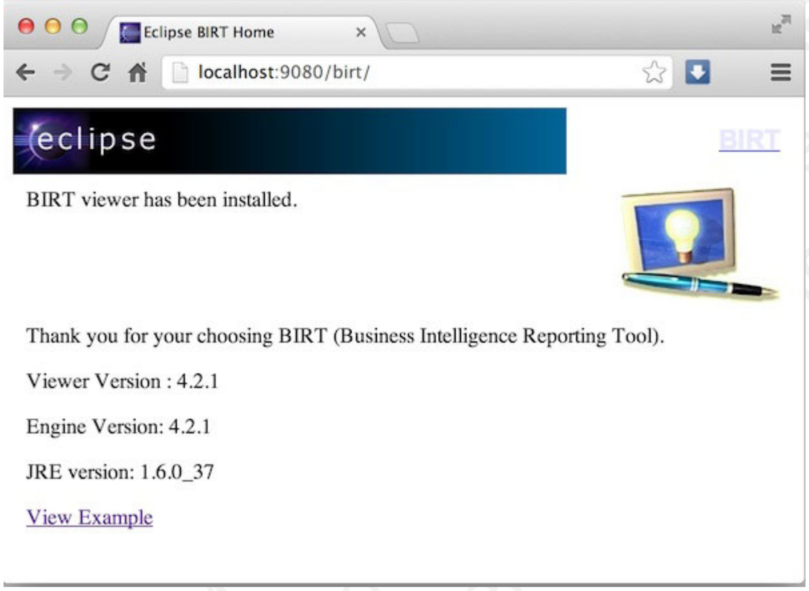
1. Verify that your WebSphere Application Server Liberty Profile instance is not running.
2. Go to your WebSphere Application Server Liberty Profile folder and create two folders as follows:
   - apps
   - libs
3. Locate the jdbc connector driver that you are using and copy it to the libs folder.
4. 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 following these steps.
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 definition with a privateLibrary definition that is 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.



BIRT viewer has been installed.

Thank you for your choosing BIRT (Business Intelligence Reporting Tool).

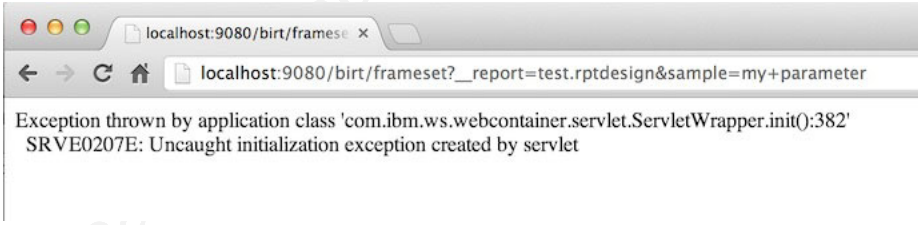
Viewer Version : 4.2.1

Engine Version: 4.2.1

JRE version: 1.6.0_37

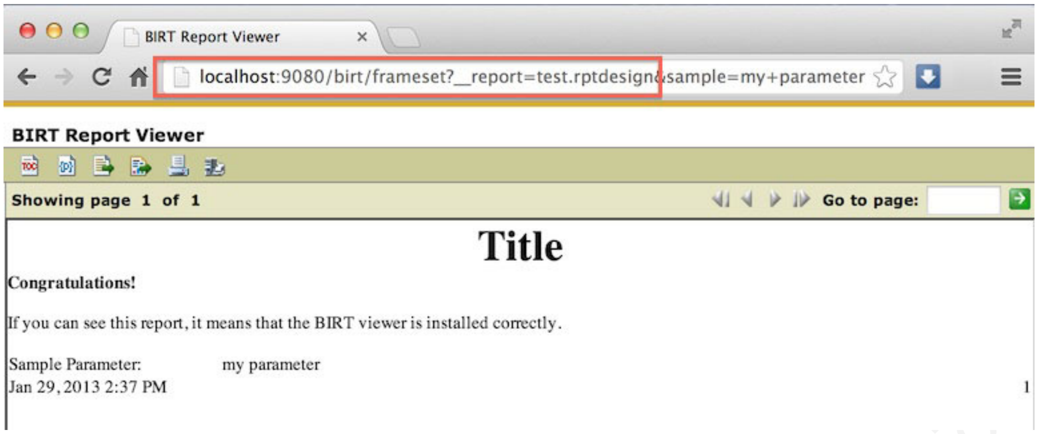
View Example

15. Click **View Example** link.
16. If you see the following error message, refresh your page.


localhost:9080/birt/framese
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter

Exception thrown by application class 'com.ibm.ws.webcontainer.servlet.ServletWrapper.init():382'
SRVE0207E: Uncaught initialization exception created by servlet

17. The BIRT Viewer sample report appears.



BIRT Report Viewer
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter
BIRT Report Viewer
Showing page 1 of 1
Go to page:

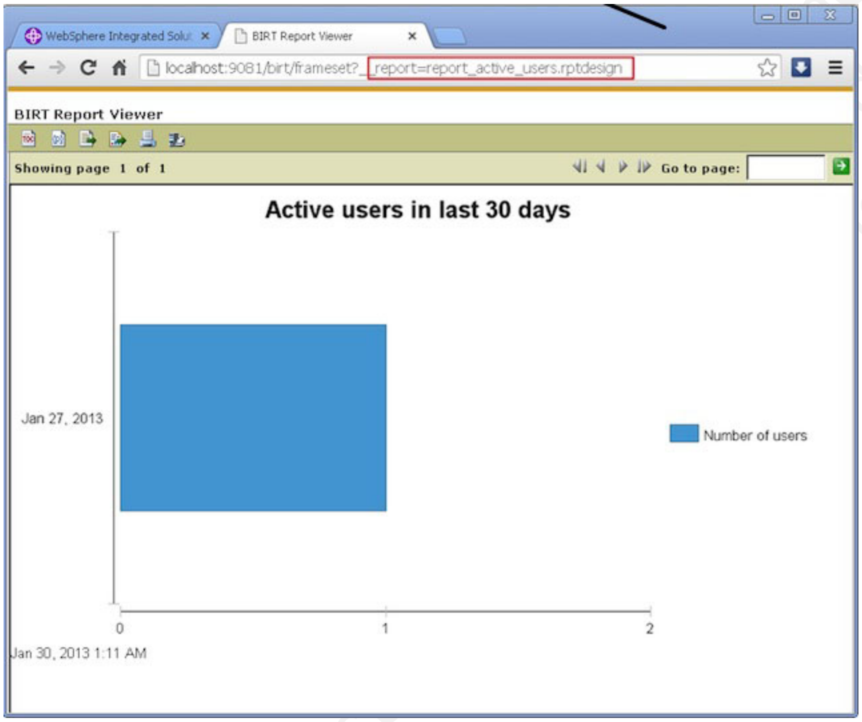
Title

Congratulations!

If you can see this report, it means that the BIRT viewer is installed correctly.

Sample Parameter: my parameter
Jan 29, 2013 2:37 PM
1

Note `test.rptdesign` in the page URL. You can replace this text with the name of other **rptdesign** files, as shown here for example:



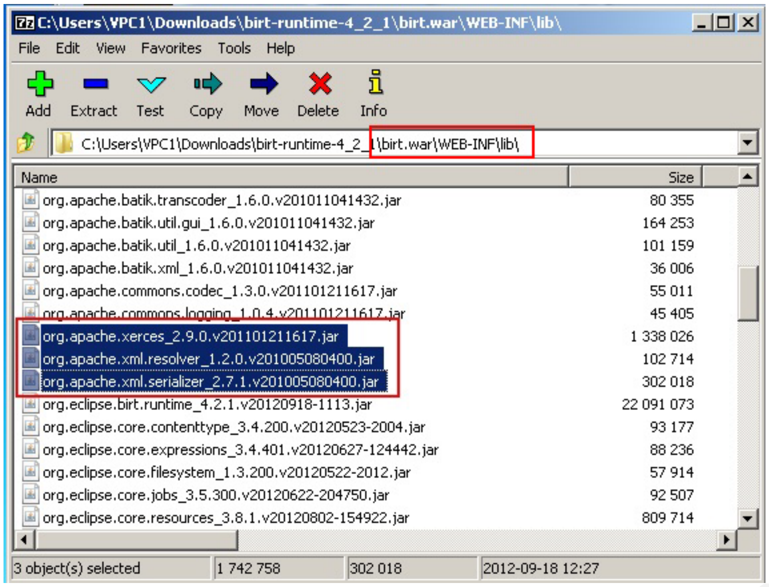
WebSphere Integrated Solut
BIRT Report Viewer
localhost:9081/birt/frameset?_report=report_active_users.rptdesign
BIRT Report Viewer
Showing page 1 of 1
Go to page:
Active users in last 30 days
Jan 27, 2013
Number of users
0
1
2
Jan 30, 2013 1:11 AM

Installing BIRT on WebSphere Application Server Full Profile

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`



C:\Users\VPC1\Downloads\birt-runtime-4_2_1\birt.war\WEB-INF\lib\

Name	Size
org.apache.batik.transcoder_1.6.0.v201011041432.jar	80 355
org.apache.batik.util.gui_1.6.0.v201011041432.jar	164 253
org.apache.batik.util_1.6.0.v201011041432.jar	101 159
org.apache.batik.xml_1.6.0.v201011041432.jar	36 006
org.apache.commons.codec_1.3.0.v201101211617.jar	55 011
org.apache.commons.logging_1.0.4.v201101211617.jar	45 405
org.apache.xerces_2.9.0.v201101211617.jar	1 338 026
org.apache.xml.resolver_1.2.0.v201005080400.jar	102 714
org.apache.xml.serializer_2.7.1.v201005080400.jar	302 018
org.eclipse.birt.runtime_4.2.1.v20120918-1113.jar	22 091 073
org.eclipse.core.contenttype_3.4.200.v20120523-2004.jar	93 177
org.eclipse.core.expressions_3.4.401.v20120627-124442.jar	88 236
org.eclipse.core.filesystem_1.3.200.v20120522-2012.jar	57 914
org.eclipse.core.jobs_3.5.300.v20120622-204750.jar	92 507
org.eclipse.core.resources_3.8.1.v20120802-154922.jar	809 714

3 object(s) selected | 1 742 758 | 302 018 | 2012-09-18 12:27

Figure 190. 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 parent 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 753 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].rtpdesign` represents one of the following files:

```
report_active_users.rptdesign
report_daily_hits.rptdesign
report_daily_visits.rptdesign
report_environment_usage.rptdesign
report_license_total_device_count.rptdesign
report_new_devices.rptdesign
report_notification_messages_per_day.rptdesign
report_notification_messages_per_source.rptdesign
```

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" ...>
    <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>ILYN</value>
      </ex-property>
      <ex-prperty>
        <name>disabledContentBidiFormatStr</name>
      </ex-property>
    </list-property>
    <property name="odaDriverClass">WLREPORT_DRIVER_CLASS</property>
```

```
      <property name="odaURL">WLREPORT_JDBC_URI</property>
      <property name="odaUser">WLREPORT_DBUSERNAME</property>
      <encrypted-property name="odaPassword" encryptionID="base64">
        WLREPORT_DBPASSWORD_BASE64
      </encrypted-property>
    </oda-data-source>
  </data-sources>
```

5. Make sure that BIRT Viewer application is installed and running on your application server
6. To view or edit a BIRT Report, go to the path `http://your-server/birt/frameset?__report=[report name].rptdesign.`, where `[report name].rtpdesign` represents one of the following files:
   - `report_active_users.rptdesign`
   - `report_daily_hits.rptdesign`
   - `report_daily_visits.rptdesign`
   - `report_environment_usage.rptdesign`
   - `report_license_total_device_count.rptdesign`
   - `report_new_devices.rptdesign`
   - `report_notification_messages_per_day.rptdesign`
   - `report_notification_messages_per_source.rptdesign`

BIRT in Eclipse

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...** dropbox, 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:

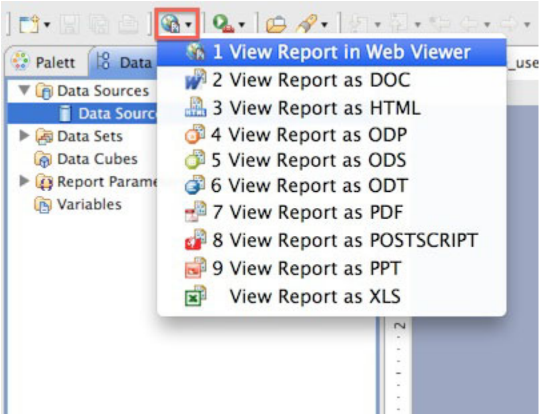
With BIRT installed in Eclipse, you can view reports through the Eclipse interface.

About this task

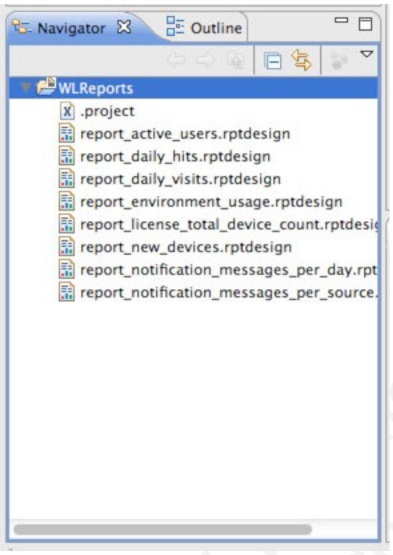
To view BIRT reports in Eclipse, follow these steps:

Procedure

1. Click the black arrow next to **View Report**.

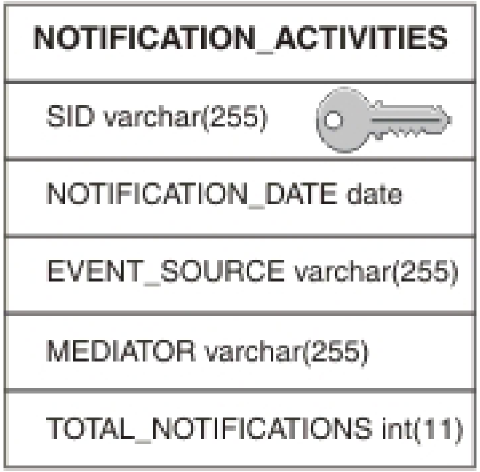


2. Select the output format for your report
3. View the report.


Navigator
Outline
WLReports
.project
report_active_users.rptdesign
report_daily_hits.rptdesign
report_daily_visits.rptdesign
report_environment_usage.rptdesign
report_license_total_device_count.rptdesi
report_new_devices.rptdesign
report_notification_messages_per_day.rpt
report_notification_messages_per_source.

Notification reports database schema

IBM Worklight uses a database schema to store the notification reports data derived from the raw data.



NOTIFICATION_ACTIVITIES

SID varchar(255)

NOTIFICATION_DATE date

EVENT_SOURCE varchar(255)

MEDIATOR varchar(255)

TOTAL_NOTIFICATIONS int(11)

Figure 191. 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.

IBM Tealeaf CX integration

IBM 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 MobileFirst 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.
- Evaluate 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.

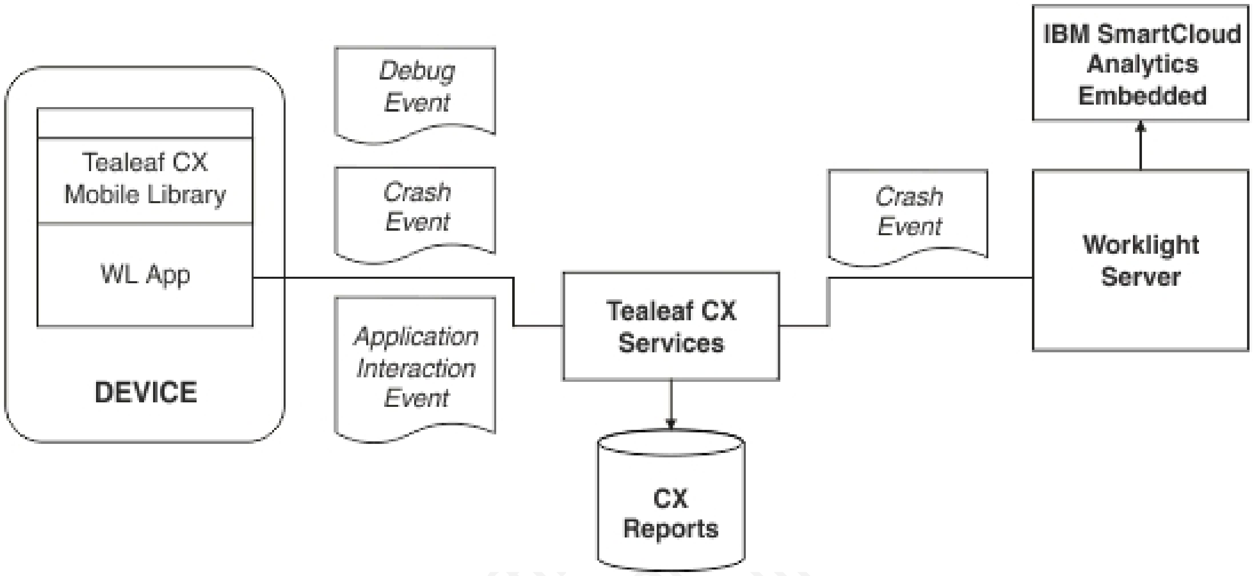


Figure 192. A possible configuration showing IBM Tealeaf CX Mobile service collecting many client session events, and forwarding only crash events to Worklight Server.

Accumulating data on IBM Tealeaf CX Mobile and IBM SmartCloud Analytics Embedded

You can configure client applications to send data to both IBM Tealeaf CX Mobile and IBM SmartCloud Analytics Embedded (operational analytics) servers.

To collect data on both IBM Tealeaf CX Mobile server and operational analytics server, client applications must be configured to send `WL.Analytics` (client session) data to the IBM Tealeaf CX Mobile server URL. Calls to the `logActivity` method, as defined in the WL.Client class, and `WL.Server.logActivity` (app activity) are never forwarded to IBM Tealeaf CX Mobile, and continue to accumulate data in the operational analytics server and in the raw reports database, if configured in each.

To reset an IBM Worklight application to send analytics data to IBM Tealeaf CX Mobile requires only one step, which is to call the `enable` method, as defined in the WL.Analytics `(url: "http://yourTealeafServer.com")`.

If the client application is already deployed and active among your users, you must call: the `restart` method, as defined in the WL.Analytics `(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 IBM Tealeaf CX server to also display in the Analytics tab in the IBM Worklight Console, the IBM Tealeaf CX Mobile server must be configured with an appropriate pre-processing script so that it can forward to IBM 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 IBM SmartCloud Analytics Embedded.

You can use the IBM Tealeaf CX Mobile client side libraries directly, instead of through the `WL.Analytics` API. Follow the IBM 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 IBM Tealeaf CX Mobile libraries directly. Do not use the `WL.Analytics` API in this configuration.

Note: In this configuration, neither crash capture nor `WL.Analytics.log` data is indexed or searchable on IBM SmartCloud Analytics Embedded. The IBM Worklight application developer now has full control of the client IBM Tealeaf CX Mobile integration.

Troubleshooting IBM Tealeaf CX integration

Find solutions to problems with IBM Tealeaf CX integration.

Table 188. IBM Tealeaf CX integration troubleshooting guidelines

Problem	Actions to take
`WL.Analytics.log` data is not searchable in the Analytics tab on the IBM Worklight Console.	The `log` method as defined in the WL.Analytics class is not sent until IBM Worklight authentication channels between the client and server are open. • Ensure that client apps have successfully authenticated with Worklight Server. • For WL applications that do not use authentication, the initialization sequence must be complete, which can be achieved by setting `connectOnStartup:true` in `initOptions.js`, making an adapter call. • Ensure that `WL.Analytics` is ready before calling the `log` method as defined in the WL.Analytics class. • If you are using Tealeaf directly and you modified the Tealeaf configuration, revert the configuration to the one generated by IBM Worklight.

For more information, see “Troubleshooting Worklight Server” on page 216

Mobile application management

The Mobile Application Management feature enables mobile operators and administrators to securely track, search, and control access to users through the mobile applications that are used on their devices, all from the Worklight Console.

The IBM Worklight Server runtime tracks devices that access your mobile infrastructure by the Worklight apps that are used by your users. Each user, whether employee, customers, suppliers, or business partners, can use several devices to access your mobile environment through one or more apps that you deployed. IBM Worklight Console now provides a view into this mapping of user to devices through the apps that are used to access your Worklight Server. Mobile operators and administrators can use the console to not only search for registered users by name, but also block access to a specific app from a specific user's device. They can also block any Worklight App that is installed on the device from connecting to the Worklight Server.

When multiple applications from the same enterprise are installed to the same device, it is desirable to disable access for all of the applications at once when the device is lost, stolen, or its security compromised. When these applications on the same device are authenticated to and routing traffic through a Worklight Server, administrators can disable access for all Worklight applications on that device.

In some cases, it might not be desirable to block access for every Worklight application that is installed on the device. Worklight application management features allow the administrator to view each individual application that is installed on a user’s device and select which applications to block access.

When a Worklight application requires a certificate from the user to authenticate, the serial number of the certificate is recorded on the Worklight Server. In addition to viewing each application installed on a device, the certificate serial number can also be viewed in the Worklight console. This feature allows administrators to revoke access to an application installed on the device by using the serial number to locate and revoke the certificate.

Worklight maintains a database table of device IDs, among other device-related metadata, to enable this feature. In addition to the device ID column in the database, a status column is also kept. The possible status values are:

- active
- lost
- stolen
- expired (the device has not connected to this Worklight server in 90 days) - configurable
- disabled

When a Worklight application from a device attempts to connect through the Worklight Server, the device ID is stored in the in-memory session data on the server. This device ID is checked against the database before any further processing of the inbound message. If the status column for this device ID is any value other than `active`, a 401 forbidden is returned. If the status is `lost`, `stolen`, or `disabled`, only an administrator with access to the Worklight console or direct database access can restore the status to the `active` state.

User to device mapping and control

Starting in Worklight V6.1.0, the Worklight Server tracks the devices that access the system as part of the core Worklight database. You can now enable the user to device mapping feature, which provides the ability for mobile operators or administrators to query their mobile systems by user. A device friendly name can also be established to see the devices that are mapped to a user. Further, specific controls can be applied to a user-app-device mapping to either disable that link or reactivate that link to address common situations. For example, a user loses a device and must block all access from that device. Another example is the requirement to block access to an app across all devices, or block access to an app on a device, when a user changes departments. Reactivation is available for all of these disablement control actions.

For the user to device mapping feature to work, a security realm must exist that establishes the user identity. The user identity is then used to associate the Worklight Device ID with the user. Developers can create custom challenge handlers or specific API calls to set a device friendly name as preferred by the user, programmatically. This feature helps in querying the device by its friendly name.

The following list shows what a mobile operator or admin can do with this set of features:

- Search for a device by friendly name or search by user name.
- A matching search yields all devices that belong to that user or the single device and the associated user, along with device model and information.
- The apps that are used on the device to access this system are also displayed.

The following list shows the available actions that can be taken for a queried device:

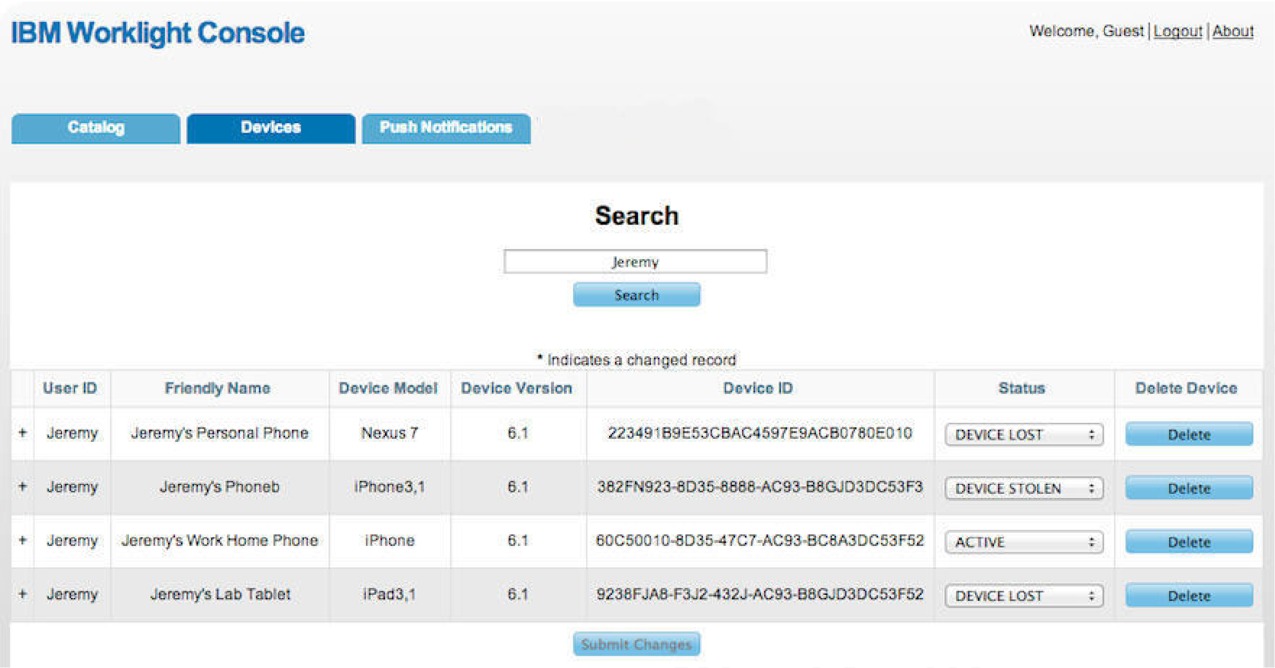
- Disable the specific device, marking the state as lost or stolen so that access from any of the apps on that device is blocked.
- Re-enable a disabled device so that access from the device to the Worklight Server is allowed.
- Disable a specific app, marking the state as disabled so that access from the specific app on that device is blocked.
- Re-enable that specific app on the device so that access from the specific app on the device to the Worklight Server is allowed.

Device access management in the Worklight Console

Since IBM Worklight V6.1.0, the console displays a new tab called `Devices`. With this tab, the IBM Worklight administrator can search for devices that accessed the Worklight Server and manage their access rights.

In the search field, devices can be searched for by either the user ID (the ID that was used to log in to the Authentication Realm), or the friendly name (a name that is associated with the device to distinguish it from other devices that share the user ID). The friendly name can be set on the client by using the client-side JavaScript APIs: `WL.Device.getFriendlyName` and `WL.Device.setFriendlyName`. For more information about the `getFriendlyName` API, see the `getFriendlyName` method, as defined in the WL.Device class. For more information about the `setFriendlyName` API, see `setFriendlyName` method, as defined in the WL.Device.

When a valid device is found, all devices that match the user ID or friendly name are listed.



Jeremy

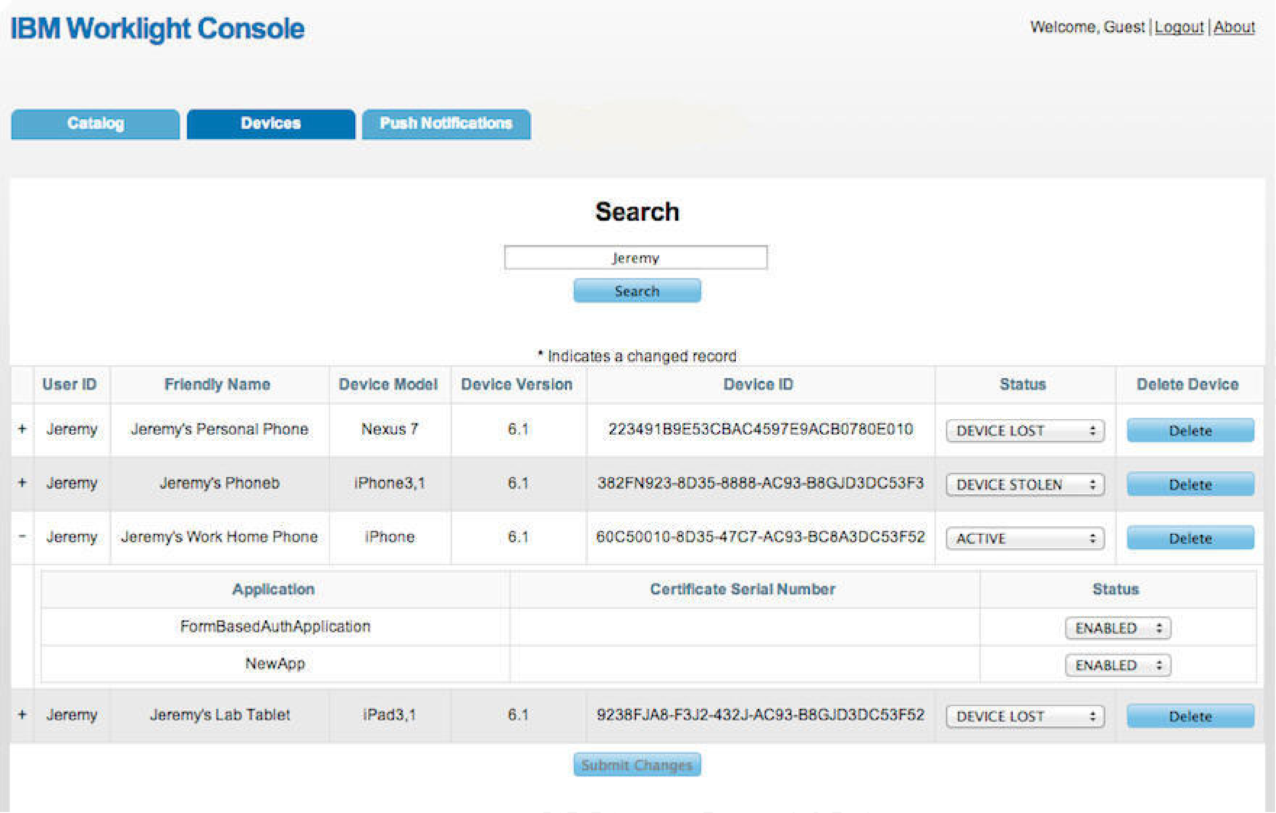
* Indicates a changed record

	User ID	Friendly Name	Device Model	Device Version	Device ID	Status	Delete Device
+	Jeremy	Jeremy's Personal Phone	Nexus 7	6.1	223491B9E53CBAC4597E9ACB0780E010	DEVICE LOST	Delete
+	Jeremy	Jeremy's Phoneb	iPhone3,1	6.1	382FN923-8D35-8888-AC93-B8GJD3DC53F3	DEVICE STOLEN	Delete
+	Jeremy	Jeremy's Work Home Phone	iPhone	6.1	60C50010-8D35-47C7-AC93-BC8A3DC53F52	ACTIVE	Delete
+	Jeremy	Jeremy's Lab Tablet	iPad3,1	6.1	9238FJA8-F3J2-432J-AC93-B8GJD3DC53F52	DEVICE LOST	Delete

Figure 193. User or friendly name search

The `Status` column contains the current access rights of the device. Any device with the column marked as `DEVICE STOLEN`, `DEVICE LOST`, or `DEVICE DISABLED` is not allowed to access the Worklight Server. The `DEVICE EXPIRED` status is used only for licensing purposes. Upon successful connection to the server, any device with the status marked as `DEVICE EXPIRED` is allowed to access the Worklight Server and its status is changed to `ACTIVE`. For more information about licensing, see “License Tracking report” on page 1012.

Clicking the + icon in the column shows a list of all applications that this device accessed.



	User ID	Friendly Name	Device Model	Device Version	Device ID	Status	Delete Device
+	Jeremy	Jeremy's Personal Phone	Nexus 7	6.1	223491B9E53CBAC4597E9ACB0780E010	DEVICE LOST	Delete
+	Jeremy	Jeremy's Phoneb	iPhone3,1	6.1	382FN923-8D35-8888-AC93-B8GJD3DC53F3	DEVICE STOLEN	Delete
-	Jeremy	Jeremy's Work Home Phone	iPhone	6.1	60C50010-8D35-47C7-AC93-BC8A3DC53F52	ACTIVE	Delete

Application	Certificate Serial Number	Status
FormBasedAuthApplication		ENABLED
NewApp		ENABLED

	User ID	Friendly Name	Device Model	Device Version	Device ID	Status	Delete Device
+	Jeremy	Jeremy's Lab Tablet	iPad3,1	6.1	9238FJA8-F3J2-432J-AC93-B8GJD3DC53F52	DEVICE LOST	Delete

Figure 194. List of applications that are accessed by a device

Each row in the table contains the name of the application, the certificate serial number for this device-application pair (if enabled), and a status menu that is used to disable an application's access to the Worklight Server for this device.

Enabling the device access management features

All devices that access the Worklight Server are recorded in the Worklight database without any additional configurations. However, Worklight does not enforce the device access settings that are set from the Worklight Console unless you enable a property on the Worklight Server.

About this task

More processing is required on the Worklight Server when this property is enabled to enforce access management on devices. Appropriate performance testing must be done before production to measure how enabling this feature impacts the server's performance.

Procedure

1. Set the `wl.device.enableAccessManagement=true` property on the Worklight Server (this value is `false` by default). The `wl.device.tracking.enabled=true` property must also be set (this value is `true` by default).
2. Capture the `UserID`. The user ID is recorded for the device automatically when the user logs in to an authentication realm that is marked as `isInternalUserID`. The following example shows a sample authentication configuration file:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<tns:loginConfiguration xmlns:tns="http://www.worklight.com/auth/config"
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
<!-- Licensed Materials - Property of IBM
     5725-G92 (C) Copyright IBM Corp. 2006, 2013. All Rights Reserved.
     US Government Users Restricted Rights - Use, duplication or
     disclosure restricted by GSA ADP Schedule Contract with IBM Corp. -->

<securityTests>
  <customSecurityTest name="DummyAdapter-securityTest">
    <test isInternalUserID="true" realm="SampleAppRealm" />
  </customSecurityTest?
</securityTests>

<realms>
  <realm loginModule="StrongDummy" name="SampleAppRealm">
    <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
  </realm>
</realms>

<loginModules>
  <loginModule name="StrongDummy">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
  </loginModule>
</loginModules>

</tns:loginConfiguration>
```

Since a security test can include several realms that require a user ID, only the realm that has the isInternalUserID property is recorded for the device in the Worklight database. For a mobileSecurityTest, the realm that is set by the testUser element is used. For more information about security tests, see “Security Tests” on page 605.

If the user is authenticated through the UserCertificateAuthenticator, the serial number that is generated for the certificate that is sent to the device is automatically saved in the Worklight database. For more information about the UserCertificateAuthenticator and serial number, see “User certificate authentication” on page 994.

Performance implications for the server

You must consider two questions when you measure the Mobile Application Management feature and its impact on performance.

1. Does Worklight save information about a device when it accesses the server?
2. Does Worklight enforce access rights when a device tries to access the server?

Saving device information

The Worklight administrator can control whether the server saves device information to the internal database when a device connects to the Worklight Server. This behavior is controlled by the following flag in the worklight.properties file:

```
wl.device.tracking.enabled=true
```

When this flag is enabled, the Worklight server attempts to store information about the device each time a device begins a new session with the server. In terms of performance, this behavior results in a potential database write each time that a device starts a new session.

Note: This flag is enabled by default in production, and is used for license tracking. Do not disable this flag unless you fully understand the implications. For more information about licensing, see “License tracking” on page 1010.

Enforcing access rights

The Worklight Server tries to save the device information only on the first request of a session from the device. However, Worklight must enforce access rights on every request that is made to the server from the device. This behavior ensures that the rights that are set by the Worklight administrator take effect immediately. This feature can be controlled by the following flag in the `worklight.properties` file:

```
wl.device.enableAccessManagement=true
```

From a performance perspective, this behavior results in an extra database read that occurs each time that the device tries to access a resource on the server. The performance hit for the read is smaller than the write for saving device information. Administrators must consider the fact that this read occurs every time that a device tries to connect to the server. When this flag is disabled, the administrator can still view the devices in the database from the Worklight console. However, they cannot block access from the device to the Worklight Server.

Space limitations for the database

Database administrators must consider how enabling the Mobile Application Management feature can affect the Worklight runtime database size. The Mobile Application Management feature does not affect the Worklight raw reports database. The following example shows a typical database row entry for a single device:

```
('db7abddf-3d5f-4b03-b3b8-f706e56e8306', 'Lucas', 'Tillman', '6.2', 'iPad2,5','2013-10-08 15:12:32',
```

For each application that the device uses, another entry is created as follows:

```
(db7abddf-3d5f-4b03-b3b8-f706e56e8306, 12, 0)
```

The size impact for each device is small. However, administrators must consider the potential size increases if their Worklight Server serves thousands of devices that use multiple applications that are hosted by the server. Devices can be deleted from the Worklight database in the Worklight console, but each device entry has a `Last Accessed` time stamp column. That time stamp gives administrators the ability to clear out old rows that are no longer being used, by creating custom queries.

Note: Database rows that contain device information are used for licensing purposes. Database administrators must not delete data from these rows if the action of deleting the data affects licensing.

User certificate authentication

Enterprises can now use X.509 client-side certificates to authenticate users, by applying a new user authentication realm to their existing security tests. This new realm is called `UserCertificateAuthRealm`. This feature allows enterprises to enroll users to their enterprise certificate authority (CA) directly from their mobile devices. The traffic between the Worklight application on the device and the Worklight Server in the enterprise can be secured over HTTPS with client-side certificates that are issued to the users as part of the initial enrollment process.

This feature is available only on the hybrid iOS and Android environments for this current release.

This feature is not supported with the FIPS 140-2 feature.

User certificate authentication overview

The User Certificate Authentication feature is a newly introduced user authentication realm in IBM Worklight V6.1 that establishes user identity with an X.509 client certificate.

With the User Certificate Authentication feature, Worklight provides a mechanism for enterprises to easily integrate their mobile infrastructure and existing public key infrastructure (PKI). With this new added function, enterprises can now authenticate users that are trying to access sensitive backend systems through mobile devices with X.509 client side certificates. Mobile clients can now present an X.509 certificate to establish a secure client identity over the transport layer security (TLS) protocol.

This feature allows enterprises to use their existing PKI to obtain full control of the user authentication and user enrollment process. An embedded PKI implementation is provided, which allows enterprises without their own PKI to quickly set it up. With the embedded PKI option, Worklight internally signs certificates and manages the validation and enrollment process.

More specifically, mobile clients are now able to present an X.509 client certificate to establish a secure connection over the transport layer security (TLS) protocol. Users are enrolled to the enterprise certificate authority (CA) directly from their device. The client certificate is then used to authenticate and establish a user identity on subsequent requests.

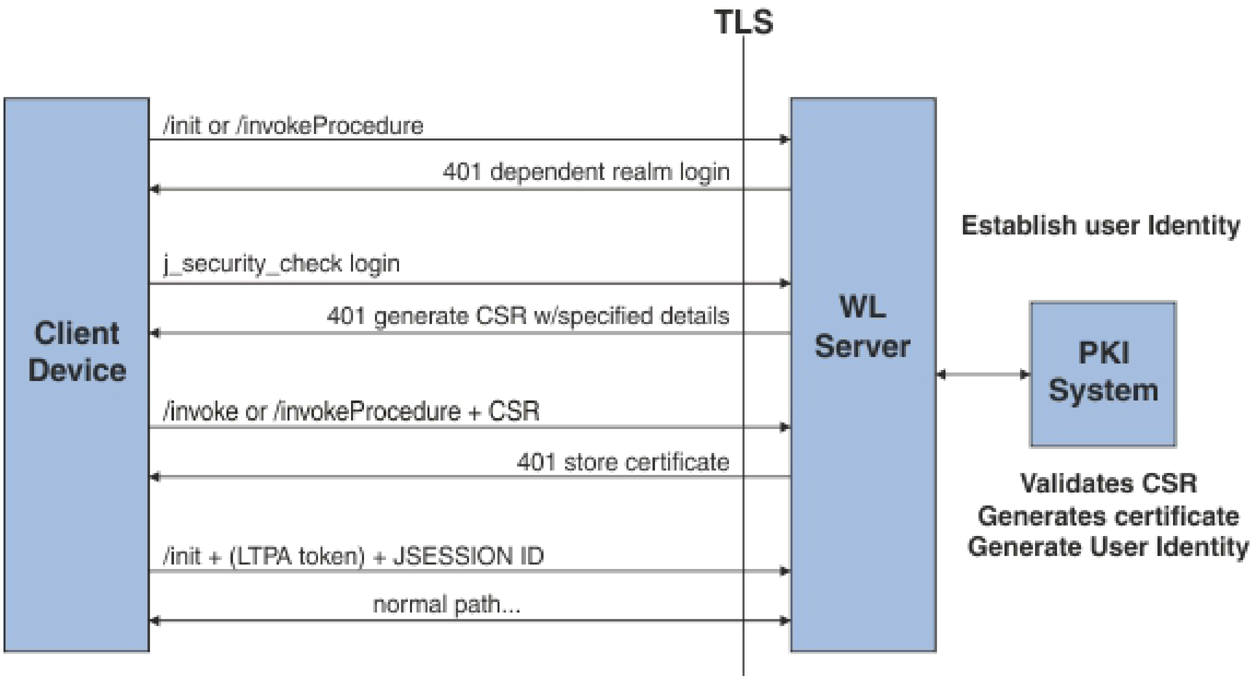
This feature is only available on hybrid iOS and Android environments for this current release.

How it works

The Worklight Server can be configured to protect an application or adapter with the user certificate authentication user realm (`UserCertificateAuthRealm`). This realm requires the use of a PKI for managing X.509 client certificates. An existing PKI can be used by implementing the PKI bridge interface that is provided for you. The PKI bridge interface serves as the bridge between Worklight and your PKI. Another option is to use the embedded PKI that is provided with this feature for testing and development purposes.

The first time a user accesses a protected application or adapter procedure from a device, the server initiates the applicable challenges and starts the user enrollment process. The user enrollment process consists of having the user enroll into the configured PKI and then provisioning the device with an X.509 certificate for future use. Users enroll into existing PKIs through the help of a dependent user authentication realm. After the user is authenticated through the dependent realm, Worklight, through the PKI, generates the client certificate and provisions the device with the certificate that is issued to the user. The server enrolls the user after successfully establishing the user identity by using one of the pre-existing login modules. This process results in an X.509 certificate that is issued to the user and installed securely on the device.

The following figure shows the user enrollment flow:

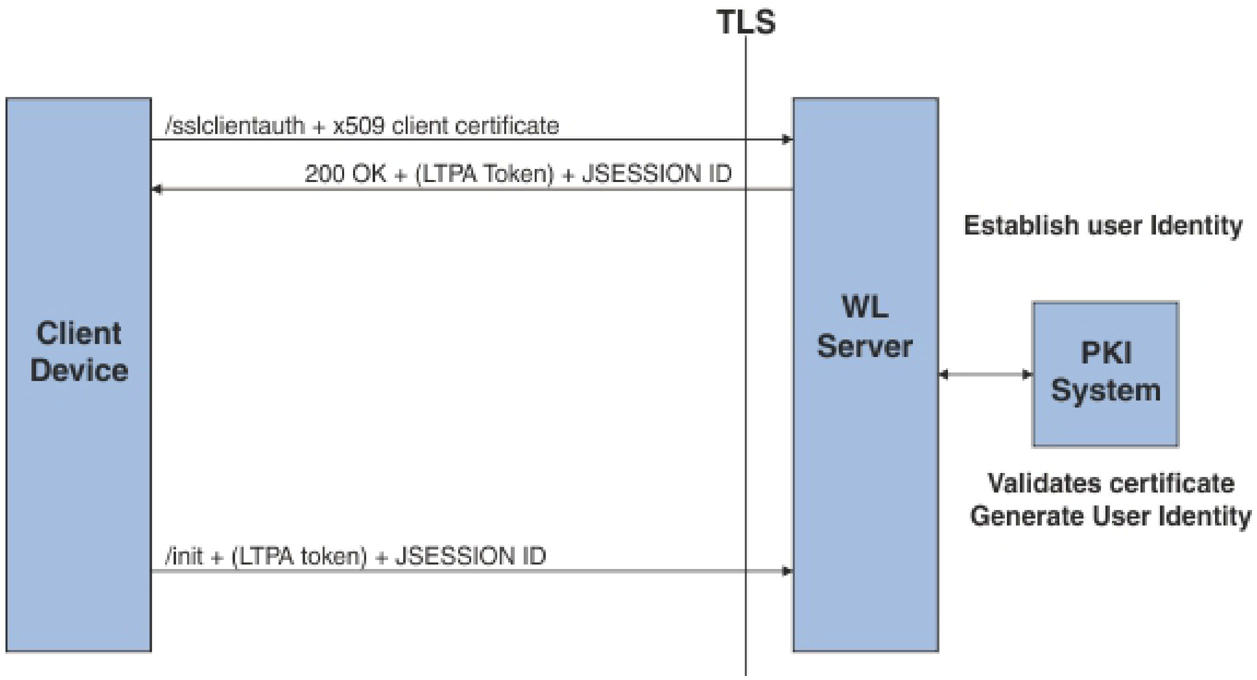


Subsequent calls from that Worklight application use this X.509 certificate to establish a secure connection over HTTPS, authenticate the user, and establish the user identity on the server. Users need to log in only once for the life of the certificate. When the certificate expires or is revoked by the PKI, the enrollment process is initiated again. You can allow user enrollment to continue, ban the user, or allow the user to log in only through the dependent realm.

Both the client and the server runtimes enforce certificate verification, ensuring that the client certificate is valid and is issued to a known user. The client certificate is valid if it is issued by a trusted CA, is not expired and is not revoked, and its validity period is current. The server also verifies the client certificate's subject against a user registry to ensure that the client certificate was issued to a known user. Support for certificate revocation lists (CRL) is provided by the underlying Java Platform, Enterprise Edition server, and JVM. For more information about how to enable CRL support in WebSphere Application Server, see SSL configurations.

Note: Not all JVMs provide CRL support.

The following figure shows the client certificate authentication flow:


TLS
Client Device
/sslclientauth + x509 client certificate
200 OK + (LTPA Token) + JSESSION ID
WL Server
Establish user Identity
PKI System
Validates certificate
Generate User Identity
/init + (LTPA token) + JSESSION ID

Protecting resources with user certificate authentication

You can protect your application or adapter procedures with the user certificate authentication user realm.

About this task

Follow the steps to configure the user certificate authentication user realm to protect your application or adapter procedure.

Procedure

1. Create a new Worklight project.
2. Create a new hybrid Worklight application.
3. Configure the challenge handlers for your dependent realm. These challenge handlers help establish the identity of the user as part of the enrollment process. For more information, see “User certificate authentication on the client” on page 1007.
4. Configure the server.
   a. Configure your WebSphere Application Server Liberty profile server. For more information, see “Configuring the Liberty profile” on page 1005.
   b. Configure the server for HTTPS. For more information, see “SSL configuration” on page 998.
   c. Configure an embedded PKI or external PKI. For more information, see “PKI bridge configuration” on page 999.
   d. Uncomment out the `wl_UserCertificateAuthRealm` realm elements in the authentication configuration and update it as needed. For more information, see “Updating the server authentication configuration” on page 1006.

5. Edit the application descriptor to specify the security test that enforces certificate authentication of the user. You can protect the application or the adapter.
6. Install the root certificate authority (CA). For more information, see "Configuring SSL between Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority" on page 125.
7. Complete the deployment to the server.
8. Install the application on the client.

What to do next

For a more comprehensive sample, see the module *Client X.509 Certificate Authentication and User Enrollment* under Category 8, *Authentication and security* in Chapter 3, "Tutorials and samples," on page 27.

User certificate authentication on the server

Both the Worklight Server and its hosting application server must be configured to use the User Certificate Authentication feature. The application server must be configured for client-side SSL. The Worklight Server must be configured with a PKI bridge and an appropriate security test to use the feature.

SSL configuration

The User Certificate Authentication feature depends on the use of SSL for authentication purposes. You can host your application only on HTTPS, unless a reverse proxy is being used.

For more information about how to configure SSL, see "WebSphere Application Server and Liberty profile requirements" on page 1005.

The User Certificate Authentication feature requires integration with a PKI. For the embedded PKI option, you are required to provide a certificate authority (CA) that can be used to generate the client X.509 certificates.

Certificates and CAs

Client certificates that are issued to the user by the User Certificate Authentication feature can be signed by a custom CA or a well-trusted CA through your PKI. Server-side certificates can be signed by either type of CA. Self-signed certificates are not supported. For more information about how to use and create an intermediate CA to sign both the server and client certificates, see the module *Client X.509 Certificate Authentication and User Enrollment* under category 8, *Authentication and security* in Chapter 3, "Tutorials and samples," on page 27.

If you encounter errors with certificates that are not signed by well-trusted CAs, see "Configuring SSL between Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority" on page 125.

Certificate chains, keystore, and trust store

You must set the server certificate as the Worklight Server's keystore. Also, set the client's certificate signing CA as part of the trust store so that the server can trust the client certificates. For more information about setting up the server with these certificates, see "WebSphere Application Server and Liberty profile requirements" on page 1005.

Note: If you use intermediate custom CAs, ensure that you concatenate the server certificate with the certificate chain. When you create the server certificate, use the following order:

```
Server certificate -> intermediate(s) in order -> trust anchor
```

The following example works in Mac OS X and Linux, and concatenates the server certificate with one intermediate CA and the trust anchor (root CA):

```
cat server/server.crt signingca/signing_ca.crt rootca/root_ca.crt > server_chain.crt
```

PKI bridge configuration

The PKI bridge is an interface between the Worklight Server and a business' public key infrastructure (PKI). Each realm definition that uses the `WorklightCertificateAuthenticator` must have a PKI bridge that is defined in its configuration.

User certificate identity versus standard Worklight user identity

The standard Worklight user identity contains basic user details and is built after a user realm is authenticated. The identity contains user name, display name, and extra attributes. The identity can be requested for each realm in a security test by authenticated resources, such as an adapter. For user certificate authentication, more details might be required, such as device ID and application name. These details are provided in the user certificate identity object that is sent to the PKI bridge.

A user certificate identity instance contains the following elements:

- Standard Worklight user identity
  - User name
  - Display name
  - Attributes
- Device ID
- Application name

Custom PKI bridge interface

A custom PKI bridge can be implemented by extending the `com.org.auth.ext.UserCertificatePKIBridge` abstract class. The API for the PKI bridge abstract class can be found at UserCertificatePKIBridge.

Embedded PKI bridge:

The embedded PKI bridge is an included PKI bridge that can be used with user certificate authentication. The embedded PKI bridge is available with the `com.worklight.core.auth.ext.UserCertificateEmbeddedPKI` class name and is configured by adding parameters to the realm definition.

The embedded PKI bridge is useful for developers without direct access to the business’ PKI during testing. Administrators that are interested in testing the user certificate authentication feature without implementing their own PKI bridge can also use the embedded PKI bridge. The embedded PKI bridge is not recommended or supported for production environments.

Requirements for use

For the embedded PKI bridge, a certificate authority (CA) certificate and private key must be available. The certificate and private key must be added to a keystore manually. The keystore must be in the PKCS #12 file format, such as a `.p12` file. A password to access the keystore can be supplied optionally in plaintext form. If the `.p12` file does not exist, cannot be read, or is supplied an invalid password, an error is thrown in the server trace. The following example shows a realm definition for `wl_userCertificateAuthRealm` with the embedded PKI:

```
<realm name="wl_userCertificateAuthRealm"
       loginModule="UserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm"
             value="WASLTPARealm" />
  <parameter name="pki-bridge-class"
             value="com.worklight.core.auth.ext.UserCertificateEmbeddedPKI" />
  <parameter name="embedded-pki-bridge-ca-p12-password"
             value="capassword" />
  <parameter name="embedded-pki-bridge-ca-p12-file-path"
             value="/opt/ssl_ca/ca.p12" />
  <parameter name="embedded-pki-bridge-organization"
             value="IBM Worklight" />
  <parameter name="embedded-pki-bridge-add-cert-extensions"
             value="true" />
</realm>
```

Configuration parameters

The following embedded PKI bridge parameters are available.

embedded-pki-bridge-ca-p12-file-path

Required

Full file path of the `.p12` file for the CA that signs user certificate requests.

embedded-pki-bridge-ca-p12-password

Optional

Password in plaintext that is used to decode the CA `.p12` file that is specified. No password is used if not specified.

embedded-pki-bridge-organization

Optional

Organization name that is added to the distinguished name (DN) inside a signed certificate (O=<organization name specified>). If not specified, no organization is added to the DN.

embedded-pki-bridge-add-cert-extensions

Optional

Add non-critical Worklight custom certificate extensions to the user certificate before it is signed. This parameter provides more details to user identity attributes on subsequent runs. These details include device id, group ID, and application name that is stored in the certificate. By default, this parameter is `false`. You can enable the parameter by using the `true` value. This parameter is not always supported and may not work for your configured server configuration. You must test this option first on your infrastructure to ensure that a certificate is not marked invalid if extensions are enabled. When this parameter is enabled, the device ID is added with the OID `1.3.6.1.4.1.2.6.256.1` and the app name is added with the OID `1.3.6.1.4.1.2.6.256.2`. These OIDs are not formally registered and may change.

embedded-pki-bridge-days-before-expire
Optional

Configure the length of time the generated certificate is valid. This setting defaults to one year (365 days).

embedded-pki-bridge-crl-uri
Optional

Configure an optional CRL for your certificate authority. If the certificate that is generated exists on a client's device and is revoked in the CRL, the client is required to generate a certificate.

External/adapter-based PKI bridge:

The adapter-based PKI bridge is an included PKI bridge that can be used with user certificate authentication. The adapter-based PKI bridge is available with the `com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI` class name, and is configured by adding parameters to the realm definition. An adapter is required for this PKI bridge to work, and must be uploaded before any user connects with this configuration. The adapter-based PKI bridge is useful if your PKI can be accessed with an adapter (such as a REST API).

Requirements for use

For the adapter-based PKI bridge, an adapter must be added in the console and the parameters for the bridge must be configured in the realm definition. The following example shows a realm definition for `wl_userCertificateAuthRealm` with the adapter-based PKI that uses an adapter that is called `PKIAdapter`:

```
<realm name="wl_userCertificateAuthRealm"
       loginModule="UserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm"
             value="WASLTPARealm" />
  <parameter name="pki-bridge-class"
             value="com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI" />
  <parameter name="adapter-pki-bridge-init-procedure"
             value="PKIAdapter.init" />
  <parameter name="adapter-pki-bridge-identity-validation-procedure"
             value="PKIAdapter.validateIdentity" />
  <parameter name="adapter-pki-bridge-csr-requirements-procedure"
             value="PKIAdapter.getCSRRequirements" />
  <parameter name="adapter-pki-bridge-csr-validation-procedure"
             value="PKIAdapter.validateCSR" />
  <parameter name="adapter-pki-bridge-certificate-generation-procedure"
             value="PKIAdapter.generateCertificate" />
  <parameter name="adapter-pki-bridge-identity-from-certificate-procedure"
             value="PKIAdapter.getIdentityFromCertificate" />
  <parameter name="adapter-pki-bridge-certificate-validation-procedure"
             value="PKIAdapter.validateCertificate" />
</realm>
```

Configuration parameters

The following adapter-based PKI bridge parameters are available.

adapter-pki-bridge-init-procedure
Required

An adapter procedure that is called to initialize the PKI bridge on each call. Requires a single parameter for the configuration that is available in the realm definition. The following example shows a sample value of this parameter:

```
{"adapter-pki-bridge-csr-validationprocedure":"
PKIBridgeAdapter.validateCSR","adapter-pki-bridge-identity-fromcertificate-
procedure":"PKIBridgeAdapter.identityFromCertificate","pkibridgeclass":"
com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI","adapterpki-
bridge-identity-validationprocedure":"
PKIBridgeAdapter.identityVerify","adapter-pki-bridge-csrrequirements-
procedure":"PKIBridgeAdapter.csrRequirements","adapter-pkibridge-
certificate-generationprocedure":"
PKIBridgeAdapter.generateCertificate","adapter-pki-bridgecertificate-
validationprocedure":"
PKIBridgeAdapter.certificateVerify","adapter-pki-bridge-initprocedure":"
PKIBridgeAdapter.init","dependent-user-authrealm":"
WASLTPARealm"}
```

adapter-pki-bridge-identity-validation-procedure

Optional

An adapter procedure that is called that allows the adapter to determine whether the user identity from the dependent realm is allowed to generate a certificate. This procedure is optional. By default, the PKI bridge always returns YES. Requires a single userIdentity parameter. The following example shows a sample value of this parameter:

```
{"deviceId":"C146B473-DA25-46A7-8A79-E8CE5E9270EE","userIdentity":
{"userName":"user@ibm.com","attributes":
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

The procedure must return an object with the following format:

```
{valid: "YES"}
```

Options for valid:

- YES - The user is allowed to generate a certificate.
- NO_USE_DEPENDENT_REALM_ONLY - The user is allowed to log in to the dependent realm, but is not allowed to generate a certificate.
- NO - The user is not allowed to log in at all, and is not allowed to generate a certificate.

adapter-pki-bridge-csr-requirements-procedure

Optional

Build a set of requirements that must be in a CSR that the client generates. This procedure is optional. By default, the CSR requirements include the commonName that is equal to the user name from the dependent realm user identity. The procedure has a single parameter that is called userIdentity with the following format:

```
{"deviceId":"C146B473-DA25-46A7-8A79-E8CE5E9270EE","userIdentity":
{"userName":"user@ibm.com","attributes":
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

This procedure must return a JSON object in the following format:

```
{ commonName: "user@ibm.com", additionalSubject: { "O": "IBM" }, additionalAttributes: {} }
```

- commonName - This attribute is a required entry that is used as the CN attribute in the CSR. This value must match a user in the user registry of the application server.

- additionalSubject - This attribute is a required JSON object that contains key/value pairs for each additional attribute that must be in the subject of the CSR, such as O for organization. If no additional attributes are required, use an empty JSON object.
- additionalAttributes - This attribute is a required JSON object that contains key/value pairs for each additional attribute that must be included in the CSR. If no additional attributes are required, use an empty JSON object.

adapter-pki-bridge-csr-validation-procedure

Optional

This procedure is called after a client sends a CSR that follows a request. It is responsible for ensuring that all of the CSR attributes that were requested in the requirements exist in the CSR. This procedure is optional. By default, the PKI bridge always returns YES. The procedure has a single parameter csr that contains a JSON object with the following format:

```
{"csr":"MIICXzCCAUcCADAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjANBgkqhkiG9w0
BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZKcf3wW2LhQ75
MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2QggqpBMFPhvBdTbS
93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABoAAwDQYJKoZIhvcNAQEFBQADggEBAHHOJbrGBCZCiDi3hXzVzji7
1euKMf8IUjGe+sfr+Sy5sfx9k
+icvKixImHCxSy0PeKp4QICSgfZxk2xQzHhYVgdeB0Uv2WT7FjPngRjAgLL1jxu7LIkEMKWgiGiJMPg
54gOx8kWuj5uE9vqpWGRK0dGuPNlnQxh50pSgZi4PhRGz2nCBF6WdQFNmHDqssijk//
CUHWbNvMTIWyuHhXEhtwkplc0dAp1b3hHBywYM9Vae9fUmfpbHDb0yvjBjCHvceRjwkoQG6ABfh9
9ucE1NWO51Rc03XqGnHKsnk16BlqSH0YpM/sVWYrmio/F9h75aNX+Sz5EhkB7t/n4301aPOo="}
```

Note: csr is the CSR in DER format and is represented in base64.

The procedure must return a JSON object with the following format:

```
{valid: "YES"}
```

or

```
{valid: "NO"}
```

Options for valid:

- YES - The CSR meets the requirements from the PKI.
- NO - The CSR does not meet the requirements from the PKI. Authentication fails.

adapter-pki-bridge-certificate-generation-procedure

Required

This procedure is responsible for requesting a certificate from the PKI and returning a certificate. This procedure is required and has one required parameter csr, which has the following format:

```
{"deviceId":"C146B473-DA25-46A7-8A79-
E8CE5E9270EE","csr":"MIICXzCCAUcCADAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjA
NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZK
cf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2Qggq
pBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABoAAwDQYJKoZIhvcNAQEFBQADggEBAHHOJbrGBCZCiDi3hXzVzji7
1euKMf8IUjGe+sfr+Sy5sfx9k
+icvKixImHCxSy0PeKp4QICSgfZxk2xQzHhYVgdeB0Uv2WT7FjPngRjAgLL1jxu7LIkEMKWgiGiJMPg
54gOx8kWuj5uE9vqpWGRK0dGuPNlnQxh50pSgZi4PhRGz2nCBF6WdQFNmHDqssijk//
CUHWbNvMTIWyuHhXEhtwkplc0dAp1b3hHBywYM9Vae9fUmfpbHDb0yvjBjCHvceRjwkoQG6ABfh9
9ucE1NWO51Rc03XqGnHKsnk16BlqSH0YpM/sVWYrmio/F9h75aNX+Sz5EhkB7t/
n4301aPOo=","userIdentity":{"userName":"lizet@us.ibm.com","attributes":
```

```
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

Note: `csr` is the CSR in DER format and is represented in `base64`.

The procedure must return a `base64` string of the X.509 certificate in DER format in a JSON object with the following format:

```
{ certificateBase64: "<BASE64 STRING OF THE X.509 CERTIFICATE>" }
```

adapter-pki-bridge-certificate-validation-procedure

Optional

This procedure is responsible for validating a user's certificate when it is first received. This procedure is optional. If it is not used, the PKI bridge always returns YES. The procedure has one parameter `certificate` that is in the same format as the procedure in the `adapter-pki-bridge-identity-from-certificate-procedure` parameter.

The procedure is required to return a JSON object that states the validity of the certificate:

```
{valid: "YES"}
```

or

```
{valid: "NO"}
```

Options for `valid`:

- `YES` - The certificate is considered valid by the PKI.
- `NO` - The certificate is not considered valid by the PKI, and the client is required to start the enrollment process over.

adapter-pki-bridge-identity-from-certificate-procedure

Required

This procedure is responsible for creating a user certificate identity from a certificate that is passed by the user. The procedure must have one parameter `certificate` with the following format:

```
{"publicKey":
{"base64":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLC
CGEBCUQAgNPZKcf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv8
1+wHaTIu2QggqpBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQAB","algorithm":"RSA"},"signature":
{"base64":"cONA8EKOQBiIKtdhAzG68pm0FMRkNfbVAIyZlttp+J9nXYmjO/
aGOEJk37oGzEPTO5uA/
eDArvQ9WF3BtzOdF9hw4j3ACJjo5oEnD7UTXbPzK2k1w3INX4cuOInLi7EJEKb
+CuO5uMy1mUOjx1aj/WaK
+E2KroFKNPyXdHAL7mwpkZO0aSYxUYYwcu8IAureMWZGps196Swk1YptboIEUSd5r3j07rBZX81B
AX5awqEx3tpbP3qpIJIK+6xoiu2tL67mKqJj9l1/Yb/
qQmUg6ouJtt9fWYUO7p1wJgUm9N0eixXftKttJ32Fp/
s0B7R72ntO9pGPrkYt8IUkzSq22Q==","algorithm":"SHA1withRSA"},"subjectUniqueId":"","version"
:1,"issuer":{"dn":"CN=Worklight Test Beta Signing CA,OU=Security Division,O=IBM
Worklight,L=Austin,ST=TX,C=US","cn":"Worklight Test Beta Signing
CA","uniqueId":""},"dn":"CN=user@us.ibm.com","cn":"user@us.ibm.com","valid":{"notBefore":
1381193593,"notAfter":
1382403193},"serialNumber":"efa7b0e3f0d9cef0","base64":"MIIDIzCCAgsCCQDvp7Dj8NnO8DA
NBgkqhkiG9w0BAQUFADCBizELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlRYMQ8wDQYDVQQHEwZ
BdXN0aW4xFjAUBgNVBAoTDUlCTSBXb3JrbGlnaHQxGjAYBgNVBAsTEVNlY3VyaXR5IERpdmlzaW9u
MSowKAYDVQQDEyFXb3JrbGlnaHQgR2FycmljayBCZXRhIFNpZ25pbmcgQ0EwHhcNMTMxMDA4M
DA1MzEzWhcNMTMxMDIyMDA1MzEzWjAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjA
NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZK
cf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2Qggq
pBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQBw40DwQo5AGIgq12EDMbry
mbQUxGQ19tUAjJmW22n4n2ddiaM79oY4QmTfugbMQ9M7m4D94MCu9D1YXcG3M50X2HDiPcA
ImOjmgScPtRNds/MraTXDcg1fhy44icuLsQkQpv4K47m4zLWZQ6PHVqP9Zor4TYqugUo0/
Jd0cAvubCmRk7RpJjFRhjBy7wgC6t4xZkamzX3pLCTVim1uggRRJ3mvePTusFlfzUEBflrCoTHe2ls/
eqkgkgr7rGiK7a0vruYqomP2XX9hv+pCZSDqi4m2319ZhQ7unXAmBSb03R6LFd+0q20nfYWn
+zQHtHvae072kY+uRi3whSTNKrbZ"}
```

Note: `base64` is the DER formatted certificate. `publicKey` is also encoded in `base64`.

The procedure must return a JSON object in the following format:

```
{ userName:"user@us.ibm.com",displayName:"",attributes:{},appID:"UserCert",deviceId:"C146B473-DA25-46A7-8
```

The goal of the JSON object that is returned is to form the original user identity of the user that is provided by the dependent realm during generation.

Note: `appId` and `deviceId` are optional in this step. If not used, use an empty string as the value.

Custom PKI bridge:

A custom PKI bridge can be implemented by extending the `com.org.auth.ext.UserCertificatePKIBridge` abstract class.

The API for the PKI bridge abstract class can be found at UserCertificatePKIBridge.

WebSphere Application Server and Liberty profile requirements

User certificate authentication uses standard SSL X.509 User Certificates, which requires the use of an SSL channel.

There are a few requirements around SSL that must be configured in order for user certificate authentication to work.

- The SSL channel for WebSphere Application Server or the Liberty profile must be configured to include the certificate authority (CA) in the trust store that is used to sign user certificates.
- The application server must be configured to allow a user certificate, but not require it. This configuration is important so that IBM Worklight can send unauthenticated challenges to the device when the device does not provide a user certificate.
- The user registry for the application server must be defined. The name that is used to authenticate a user against that user registry must match the common name (CN) in a generated user certificate.
- If you want to protect the IBM Worklight Server with the WebSphere Application Server Liberty Profile security mechanisms, you must install a fix for APAR PI10103 for Liberty Versions 8.5.5.0 and 8.5.5.1. For more information, see An IBM Worklight client cannot handle basic authentication without a WebSphere Application Server Liberty Profile fix.

Configuring the Liberty profile:

You must enable an HTTPS endpoint in WebSphere Application Server Liberty profile that uses the server's certificate, and trusts the client certificates.

Before you begin

Ensure that you understand the documentation at Enabling SSL communication for the Liberty profile. To set up the Worklight Server, see the WebSphere Application Server Liberty profile documentation about setting up SSL for the server at Liberty profile: SSL configuration attributes.

About this task

The application server requirements can be configured on the WebSphere Application Server Liberty profile in the `server.xml` file.

Procedure

1. Install a server certificate for use by the SSL channel, and configure the SSL channel.
2. Add a trust store to the configuration that contains a keystore with the CA certificate that is used to sign user certificates. Add the following element to the `server.xml` file:
```
<keyStore id="defaultTrustStore" location="trust.jks" type="JKS" password="defaultPWD" />
```
3. Enable the client authentication support by adding the `clientAuthenticationSupported="true"` attribute to the SSL element in the `server.xml` file.
4. Access the Worklight Console over SSL. You are presented with a trusted website that asks for an optional user certificate.

Updating the server authentication configuration

A requirement to enable the User Certificate Authentication feature is to configure the authentication configuration on the Worklight Server.

About this task

You must update the `authenticationConfig.xml` file to configure your server to use the User Certificate Authentication feature. User certificate authentication uses standard Worklight authentication mechanisms: authenticator and login modules. The `com.worklight.core.auth.ext.UserCertificateAuthenticator` and the `com.worklight.core.auth.ext.UserCertificateLoginModule` modules are bundled with the core Worklight Server library.

Procedure

1. From within your server configuration, open the `authenticationConfig.xml` file for editing.
2. Add a realm definition inside the `<realms>` attribute in your `authenticationConfig.xml` file.
```
<realm name="wl_userCertificateAuthRealm"
       loginModule="UserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm"
             value="<DEPENDENT REALM NAME HERE>" />
  <parameter name="pki-bridge-class"
             value="<PKI BRIDGE CLASS>" />
</realm>
```
3. Modify this realm definition by supplying your own dependent realm by specifying its name for the `dependent-user-auth-realm` parameter and a PKI bridge implementation (full Java class path) for the `pki-bridge-class` parameter. You can use the included PKI bridge classes such as embedded

(“Embedded PKI bridge” on page 999) or adapter-based (“External/adapter-based PKI bridge” on page 1001) or supply your own custom PKI bridge implementation (“Custom PKI bridge” on page 1005).

4. Add your custom parameters to this realm definition based on your PKI bridge implementation. Bundled PKI bridge implementations such as Embedded (“Embedded PKI bridge” on page 999) or Adapter-Based (“External/adapter-based PKI bridge” on page 1001) have extra required parameters that must be added.
5. Add the following login module definition, as-is, to your `<loginModules>` element in the `authenticationConfig.xml` file.

   ```
   <loginModule name="UserCertificateLoginModule">
     <className>com.worklight.core.auth.ext.UserCertificateLoginModule</className>
   </loginModule>
   ```
6. Add the `wl_userCertificateAuthRealm` realm as a test in the security test that you want to use for your application or environment.
7. Add the security test to the resource you want to protect. To protect an adapter procedure, add the `securityTest` attribute for the procedure. For more information, see “Overview of IBM Worklight adapters” on page 525. To protect an application environment, define a security test for each environment in the `application-descriptor.xml` file, by using the `securityTest="your_test_name"` property. If no security test is defined for a specific environment, only a minimal set of default platform tests are run.

   ```
   <securityTest name="your_test_name">
     <testUser realm="wl_userCertificateAuthRealm" />
     <testDeviceId provisioningType="none" />
   </securityTest>
   ```

   Note: To protect your application or adapter procedure, reference your security test in your application descriptor file.

   ```
   <iphone bundleId="com.UserCertApp" version="1.0" securityTest="your_test_name">
   ```

User certificate authentication on the client

The User Certificate Authentication feature requires little configuration on the client side. The Worklight client run time takes care of most of the heavy lifting on your behalf. There are however, a few things you need to be aware of to ensure successful and secure communication with your server.

Establishing trust

Because the User Certificate Authentication feature requires communication over HTTPS, the first thing you must ensure is that your client device trusts the server's credentials that are sent on the SSL handshake.

Each mobile platform comes with a predefined set of trusted certificate authorities (CAs) that are deemed trustworthy by the platform. Trust is easily established if your server uses a server certificate that is signed by one of these trusted CAs.

However, if your server uses a CA that is unknown to your device, you must do some extra work on the client side to establish appropriate trust. To establish trust, you must install the trust anchor certificate on the client device. The trust anchor is either the root CA, or the root certificate if you are using a self-signed certificate. For more information, see “Configuring SSL between Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority” on page 125.

Dependent user realm

The first time a user attempts to connect to the server, Worklight tries to enroll the user into the PKI and provision the device with the user certificate. To enroll the user, Worklight requires the help of a dependent user authentication realm. This behavior is all configured on the server. But you must ensure that your application has the appropriate challenge handlers that are required to handle the challenges that come from the server. The dependent realm challenge handlers do not require any additional configuration. For more information, see the appropriate section of this user documentation or getting started modules for instructions on how to write the respective challenge handlers for your dependent user realm.

iOS Group Support

User certificates are issued by default to a user on a specific application and device. Support for iOS is provided to issue a certificate to the user on a specific device and to a group of applications. The same user certificate can be shared among a group of applications that are installed on the device, allowing the user to only authenticate through a dependent realm once, and not for every application.

In this case, the user enrollment process that requires the user to log in to a dependent realm happens the first time that the user attempts to log in to the server on a particular device. After the device is provisioned with the necessary certificate, all subsequent authentications to the server from any of the Worklight applications that are designated by you use the same certificate to authenticate to the server.

To configure the sharing of user certificates among a group of applications, see "Configuring user certificate authentication for a group of applications" on page 1009.

Clearing certificates on the chain

Certificates on the client are managed by the Worklight client run time. They are installed and removed from the device as needed. However, there might be situations when you want the ability to clear the certificates that are installed on the device. For this reason, a JavaScript API is provided. The API allows the application to remove the certificates at more convenient times, like during test and development, or when the device is transferred to a new user.

The following API removes the certificate on the device for the specific application in use:

```
WL.UserAuth.deleteCertificate();
```

On iOS only, if you would like to delete the certificate that is associated with a specific group of applications, use the following API:

```
WL.UserAuth.deleteCertificate("yourGroupNameHere");
```

Security considerations

This new feature introduces a powerful and ITU-T X.509 standards-based way to authenticate users. It also introduces a password-less login mechanism. The identity is established by the Worklight client run time as part of the application that presents the certificate as part of the server-side connection. Although this behavior greatly simplifies the user experience, the following precautions must be taken by the enterprise. These precautions ensure that there is adequate protection on the device to ensure cases where the user loses the device or when the device is stolen.

1. Single user is required. The device is owned and used only by a single user and not accessible to others.
2. Device must be maintained under a device passcode lock or PIN to ensure that only the designated user can access the device and applications.

Configuring user certificate authentication for a group of applications

You can configure the User Certificate Authentication feature to issue a certificate to a user on a device for a group of applications that are protected by the user certificate authentication realm. This configuration allows a user to authenticate once and be automatically authenticated to a set of applications on the device.

About this task

You can configure the User Certificate Authentication feature to allow a group of applications to authenticate with the same X.509 client certificate. This function is supported only on iOS environments.

Procedure

1. Add a property to the `worklight.plist` worklight property list file on every application that you want to be part of the same group.

   ```
   Key: x509AccessGroup
   Value: <choose a name for your group>
   ```

2. Ensure that all applications in your group are part of the same iOS keychain-access-groups that are specified in your entitlement property list file. By default, Worklight hybrid applications are part of the `worklight.group` access group that is defined in the entitlement property file. Ensure that this group continues to be the first group in the list.
3. The application-identifier entitlement must be the same for all the applications in your group. Only applications from the same organization can share user certificates.
4. Ensure that all applications in this group are protected on the server by the same user certificate authentication realm.

Troubleshooting the User Certificate Authentication feature

Find solutions to problems with the User Certificate Authentication feature.

Table 189. User Certificate Authentication troubleshooting guidelines

Problem	Actions to take
The server is not responding even though it is accessible through the browser when it uses a certificate that is signed by a private CA.	Make sure that you can reach the Worklight Server on your device. For example, go to the Worklight Console on the device's internet browser. If you can reach it, then the most likely error is that the client is not trusting the server's certificate. The server's certificate is most likely a certificate that is signed by a private CA. To fix this problem, you must install the root CA on the device so that it is trusted. For more information, see "Establishing trust" on page 1007.

Table 189. User Certificate Authentication troubleshooting guidelines (continued)

Problem	Actions to take
Certificates that are signed by a private CA work on Android but not on iOS.	When Android is in debuggable mode, some SSL errors are ignored. This behavior gives the impression that SSL is working. Android is in debuggable mode when the APK is unsigned, or when you explicitly set it in the manifest. Worklight sets it to debuggable in the manifest by default. To make the behavior consistent, set the Android application to `debuggable:false` 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 to trust certificates that are signed by your private CA, see "Configuring SSL between Worklight Servers and clients by using certificates that are not signed by a trusted certificate authority" on page 125.
`javax.net.ssl.SSLPeerUnverifiedException` on Android or `WLSecureRequest:sendRequestToServerWithURL A connection failure occured: SSL Problem (Possible causes may include a bad/expired/self-signed certificate, clock set to the wrong date)` on iOS.	One of the certificates was not trusted. Usually it is because the server did not send the server certificate with the whole certificate chain in the right order, when it uses an intermediate CA. For more information, see "SSL configuration" on page 998. Another explanation can be that the certificate was revoked by the certificate revocation list (CRL), and the PKI did not allow the device to renew the certificate.
Authentication fails with an exception in the PKI.	There was an exception somewhere in the PKI bridge. To see more information about the exception, make sure that the Worklight Server has trace that is enabled for `com.worklight.*=all`, and search for `UserCertificate*` in the trace file. Possible reasons include a syntax or runtime error in the adapter when you use the adapter-based PKI bridge, or a configuration error in the embedded PKI.
The client certificate is expired or not yet valid.	If the certificate is expired or not yet valid, the client logs this information in the client's logs. The client then proceeds with the authentication as if it did not have a certificate. The PKI then decides whether it allows the user to renew the certificate or not. In the 'certificate not yet valid' scenario, verify that the device and the server clocks are set correctly.

License tracking

IBM Worklight is available in Enterprise (B2E) and Consumer (B2C) editions, and the license terms vary depending on which edition was sold.

License tracking is enabled by default in Worklight, which tracks metrics relevant to the licensing policy such as active client devices and installed apps.

This information helps determine if the current usage of Worklight is within the license entitlement levels and can prevent potential license violations.

Also, by tracking the usage of client devices, and determining whether the devices are active, IBM Worklight administrators can decommission devices that are no longer accessing the Worklight platform. This situation might arise if an employee has left the company, for example.

License tracking details are gathered by specifying configuration properties in JNDI, and the data that is gathered is displayed in a License Tracking report that is accessed from the Worklight Console.

Configuring your license tracking details

Administrators can set Java Naming and Directory Interface (JNDI) configuration properties to gather data that relates to license terms for devices that are accessing the IBM Worklight platform. This data can be displayed in the License Tracking report, which is accessed from the Worklight Console.

About this task

Administrators can specify the following JNDI configuration properties, which enable the administrators to gather the required data:

wl.device.decommission.when
The number of days of inactivity after which a client device is decommissioned by the device decommissioning task. The default value is 90 days.

wl.device.archiveDecommissioned.when
A value, in days, that defines when client devices that were decommissioned will be placed in an archive file when the decommissioning task is run. The archived client devices are written to a file in the Worklight Server `home\devices_archive` directory. The name of the file contains the time stamp when the archive file is created. The default value is 90 days.

wl.device.tracking.enabled
A value that is used to enable or disable device tracking in Worklight. For performance reasons, you can disable this flag when Worklight is running only Business-to-Consumer (B2C) apps. When device tracking is disabled, the license reports are also disabled and no license metrics are generated.

For more information about specifying JNDI properties, see Configuring an IBM Worklight project in production by using JNDI environment entries.

The decommissioning task is run daily, as a Worklight Server task in the background. This task performs the following actions:

- Decommissions inactive devices, based on the **wl.device.decommission.when** setting.
- Optionally, archives older decommissioned devices, based on the **wl.device.archiveDecommissioned.when** setting.
- Generates the License Tracking report.

Active client devices are those devices whose status is not decommissioned; inactive client devices have a decommissioned status.

Procedure

1. Specify the required properties as JNDI properties.

2. View the data in the License Tracking report in the Worklight Console. For more information, see “License Tracking report.”

License Tracking report

IBM Worklight provides a report that shows how many client devices are accessing the platform, and whether they are active or decommissioned. The report also provides historical data.

The License Tracking report shows the following data:

- The number of applications deployed in the Worklight Server
- The number of client devices, both active and decommissioned
- The highest number of client devices reported over the last *n* days, where *n* is the number of days of inactivity after which a client device is decommissioned.

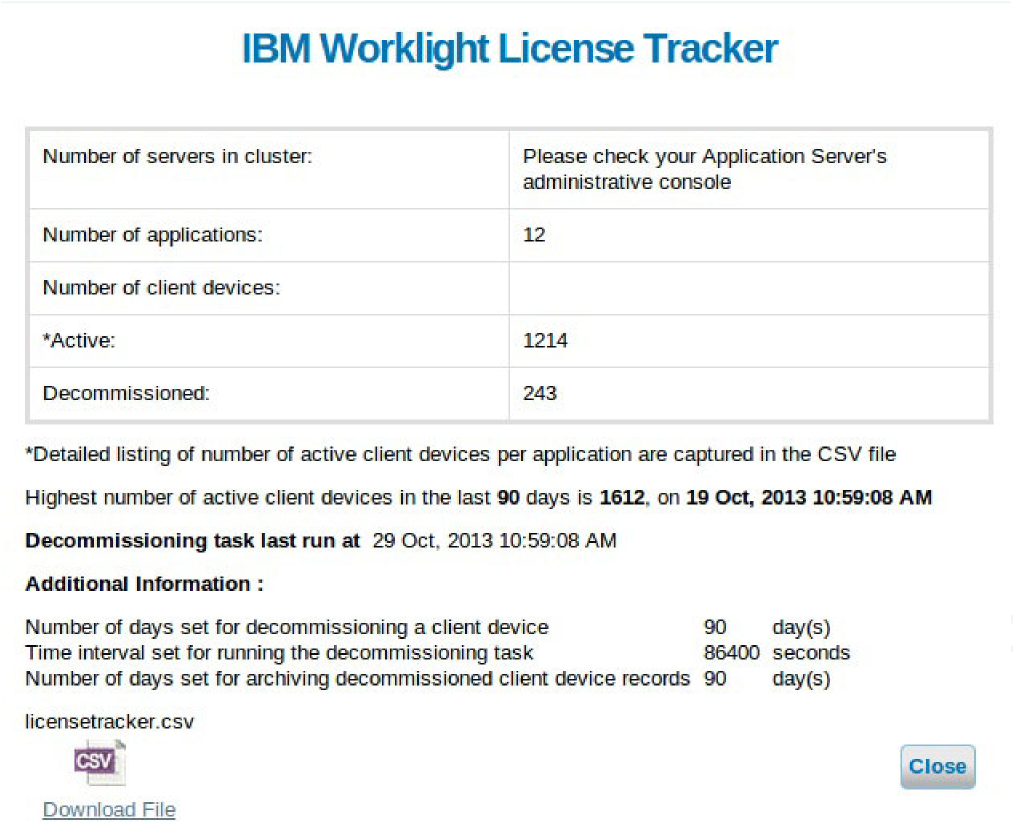
Administrators might want to analyze data further. For this purpose, the number of active client devices per application, the generated report details, and an historical listing of license metrics are captured in a CSV file that can be downloaded for further analysis.

The data is gathered by using the following JNDI configuration properties:

- **`wl.device.decommission.when`**
- **`wl.device.archiveDecommissioned.when`**
- **`wl.device.tracking.enabled`**

For more information, see Configuring your license tracking details.

To access the License Tracking report, click the **Badge** icon in the lower left corner of the Worklight Console (🏅). The License Tracking report is displayed:



IBM Worklight License Tracker

Number of servers in cluster:	Please check your Application Server's administrative console
Number of applications:	12
Number of client devices:	
*Active:	1214
Decommissioned:	243

*Detailed listing of number of active client devices per application are captured in the CSV file

Highest number of active client devices in the last **90** days is **1612**, on **19 Oct, 2013 10:59:08 AM**

Decommissioning task last run at 29 Oct, 2013 10:59:08 AM

Additional Information :

Number of days set for decommissioning a client device	90	day(s)
Time interval set for running the decommissioning task	86400	seconds
Number of days set for archiving decommissioned client device records	90	day(s)

licensetracker.csv

To save key details from the License Tracking report in a CSV file, click the **CSV** icon at the lower left corner of the report.

Chapter 13. Integrating with other IBM products

To integrate IBM Worklight with other IBM products, you implement adapters and authentication features.

This topic is intended for developers and administrators who want to understand the various integration options available with IBM Worklight, specifically concerning IBM Endpoint Manager for Mobile Devices, IBM WebSphere Cast Iron, IBM WebSphere DataPower, and IBM Security Access Manager.

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 195 on page 1016 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-end systems and other Internet or intranet sources. IBM Worklight Enterprise Edition and IBM Worklight Consumer Edition provide capabilities to integrate 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.

Item	Description
A	App
D	Device
N	Network
I/i	Internet or intranet
WL	IBM Worklight
EBE	Existing back ends
I	Other Internet sources

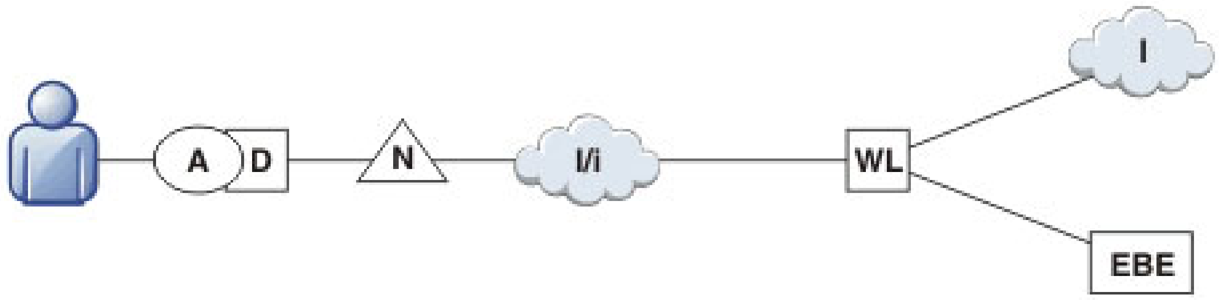


Figure 195. Overall Topology

Figure 196 shows where these products fit within the typical IBM Worklight topology diagram shown in Figure 195.

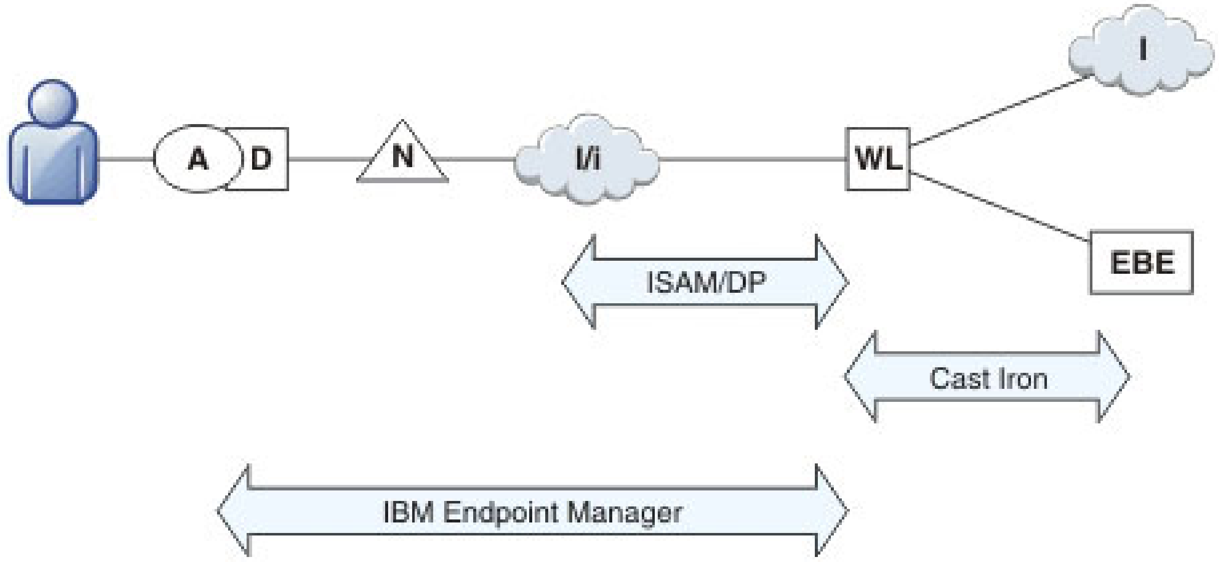


Figure 196. Integration Points

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 197 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.

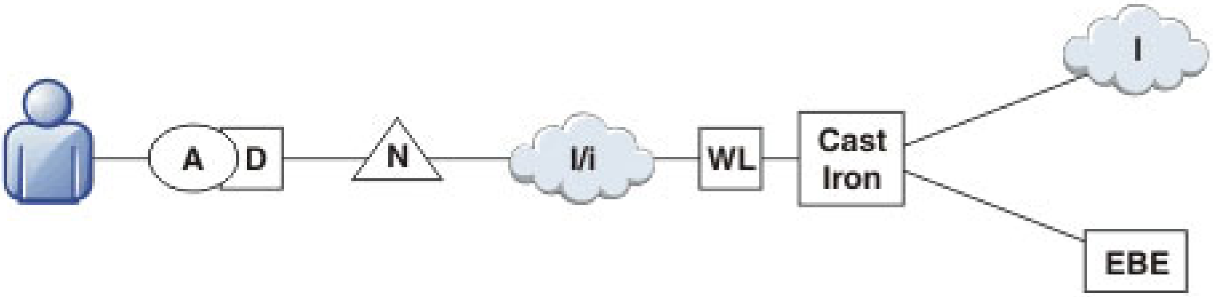


Figure 197. Integration with Cast Iron

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 27.

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 198, and follow the gateway pattern.

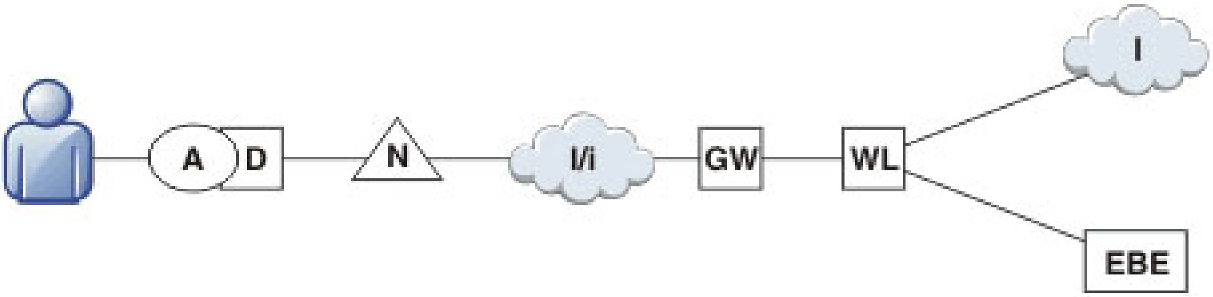


Figure 198. Integration with reverse proxy

The gateway icon (GW) represents a reverse proxy such as WebSphere DataPower, or IBM Security Access Manager. 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 199.

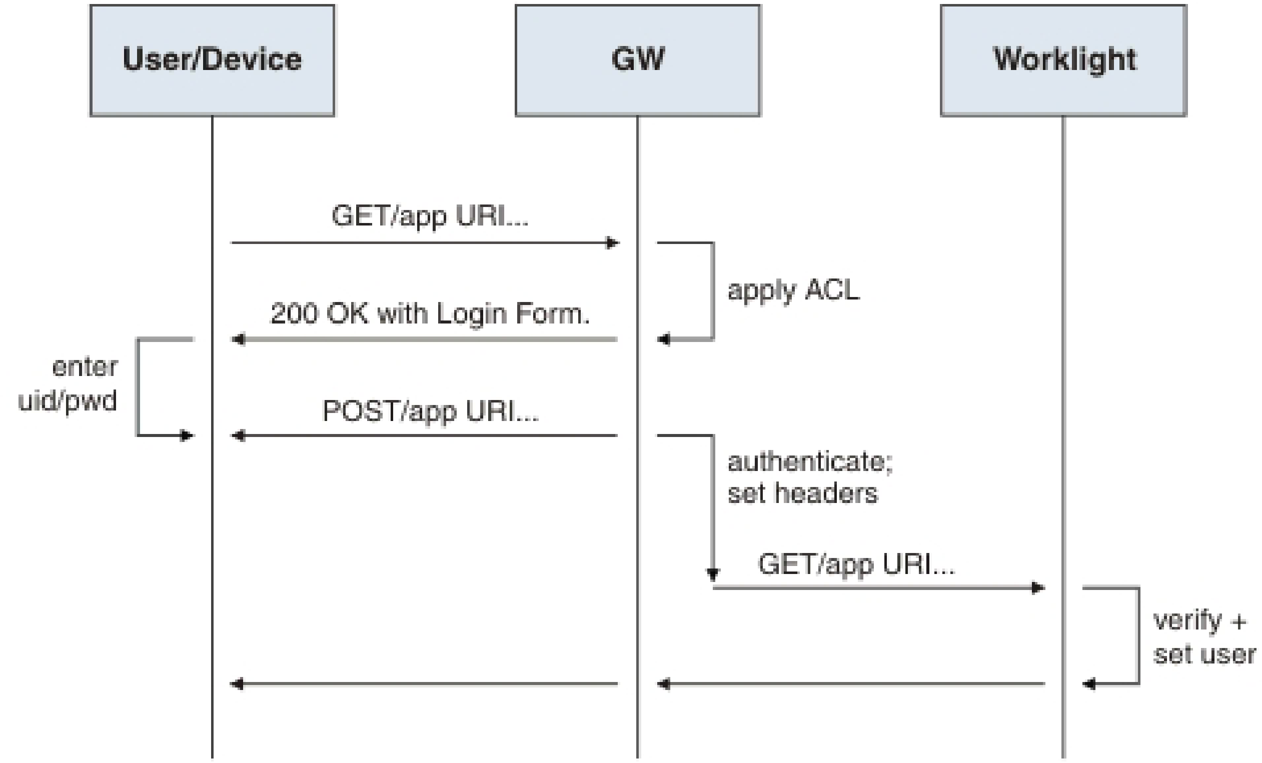


Figure 199. Authentication flow

This configuration was tested with DataPower and IBM Security Access Manager 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

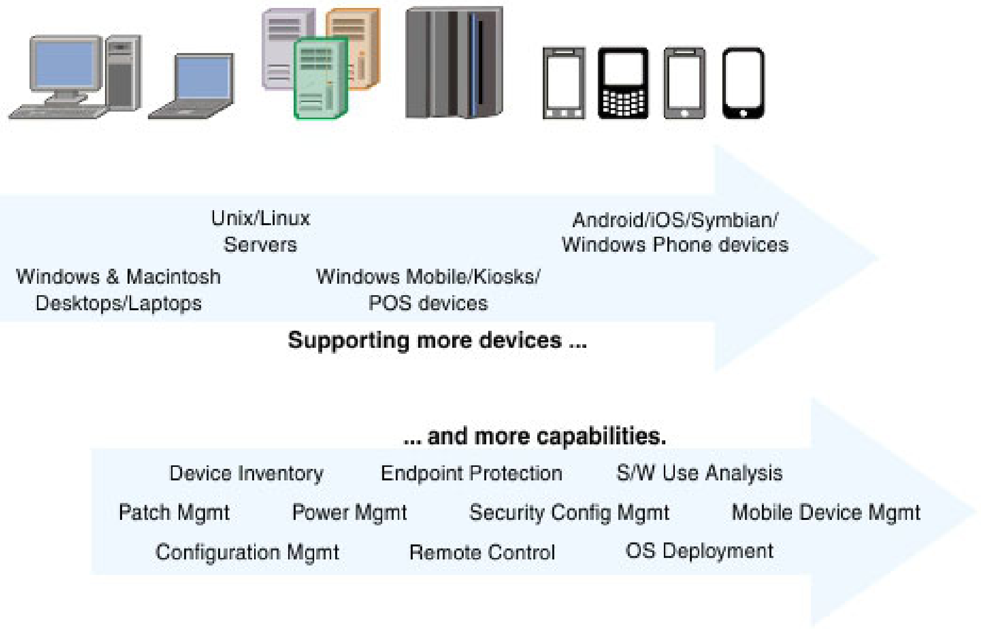
Use of LTPA-based authentication to log in to IBM Worklight through a reverse proxy.

- 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 – Enterproid 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.

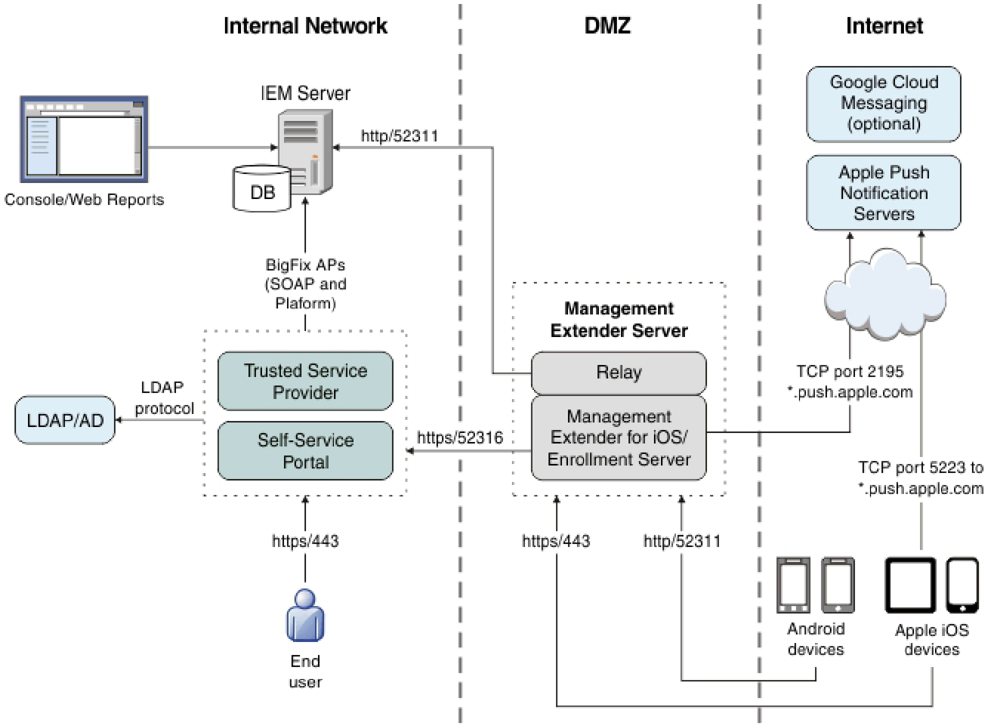
- PIM (Personal Information Manager) – NitroDesk TouchDown is a PIM product that allows enterprise data such as email to be secured separately in a BYOD Android environment.
- Support for SAFE (Samsung's proprietary APIs that allow more security than standard Android).

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.


Internal Network
DMZ
Internet
Console/Web Reports
IEM Server
DB
http/52311
Google Cloud Messaging (optional)
Apple Push Notification Servers
BigFix APs (SOAP and Plaform)
Management Extender Server
Relay
Management Extender for iOS/ Enrollment Server
Trusted Service Provider
Self-Service Portal
LDAP protocol
LDAP/AD
https/52316
TCP port 2195 *.push.apple.com
TCP port 5223 to *.push.apple.com
https/443
https/443
http/52311
Android devices
Apple iOS devices
End user

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 200.

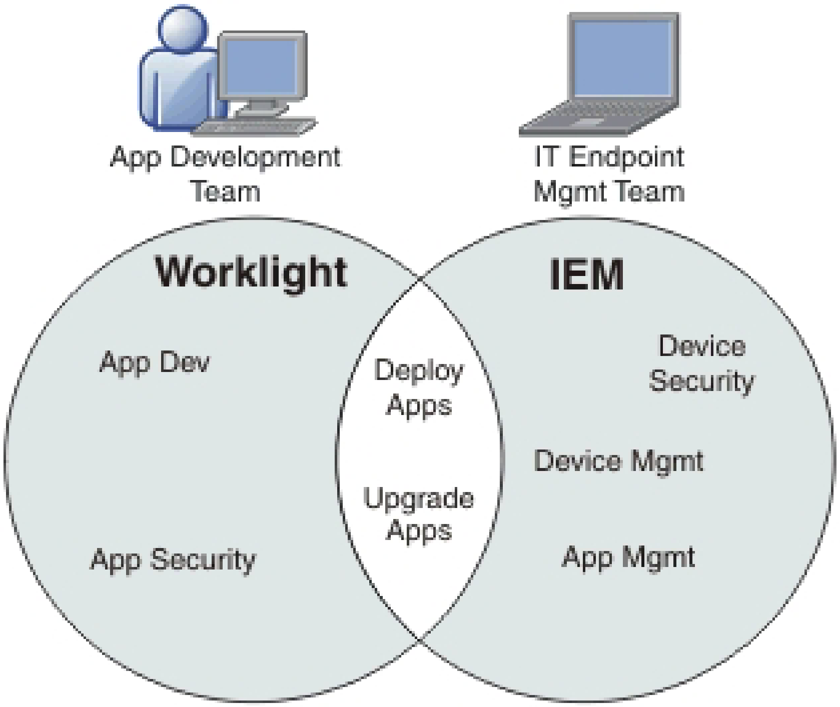


Figure 200. IBM Worklight and IBM Endpoint Manager management capabilities

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.

Using WebSphere DataPower as a push notification proxy

IBM WebSphere DataPower can be used as a gateway for outbound connections to facilitate monitoring and routing. IBM Worklight makes outbound connections to notification mediators in order to push notifications for mobile applications. You can set up DataPower to act as a push notification proxy for Worklight mobile applications.

About this task

IBM WebSphere DataPower SOA Appliances are built for simplified deployment and hardened security, bridging multiple protocols, and performing conversions at wire speed. These capabilities help an organization to achieve and maintain its security and operational polices.

DataPower can act as a reverse proxy and security gateway for handling inbound traffic into an enterprise. In addition, in a situation where corporate policy mandates that all outbound connections must be made through a gateway to facilitate monitoring and routing, DataPower can also be used as a gateway for such a requirement.

IBM Worklight makes outbound connections to a notification mediator, APNS (Apple Push Notification Service) or GCM (Google Cloud Messaging servers), in order to push notifications for mobile applications. DataPower can act as a proxy between IBM Worklight Server and APNS or GCM.

Procedure

- For both APNS and GCM, you must configure both DataPower and the Worklight Server.
- For GCM, there are two possible DataPower configurations that would enable it to act as a GCM proxy for Worklight: a TCP proxy configuration and a web application firewall configuration.
- For more information, and detailed step-by-step instructions, see the developerWorks article Using WebSphere DataPower as a push notification proxy for Worklight mobile applications.

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 14. 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 15. Glossary

This glossary provides terms and definitions for the IBM Worklight software and products.

The following cross-references are used in this glossary:

- *See* refers you from a nonpreferred term to the preferred term or from an abbreviation to the spelled-out form.
- *See also* refers you to a related or contrasting term.

For other terms and definitions, see the IBM Terminology website (opens in new window).

“A” “B” on page 1030 “C” on page 1030 “D” on page 1031 “E” on page 1032 “F” on page 1032 “G” on page 1032 “H” on page 1033 “I” on page 1033 “K” on page 1033 “L” on page 1033 “M” on page 1034 “N” on page 1034 “P” on page 1034 “R” on page 1035 “S” on page 1035 “T” on page 1036 “V” on page 1036 “W” on page 1037 “X” on page 1037

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. See also Company Hub.

Application Center installer
An application that lists the catalog of available applications in the Application Center. The Application 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. 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.
2. A server-side component that issues a sequence of challenges on the server side and responds on the client side. See also challenge handler.

B

Base64
A plain-text format that is used to encode binary data. Base64 encoding is commonly used in User Certificate Authentication to encode X.509 certificates, X.509 CSRs, and X.509 CRLs. See also DER encoded, PEM encoded.

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

CA See certificate authority.

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. See also certificate authority.

certificate authority (CA)
A trusted third-party organization or company that issues the digital certificates. The certificate authority typically verifies the identity of the individuals who are granted the unique certificate. See also certificate.

certificate authority enterprise application
A company application that provides certificates and private keys for its client applications.

certificate revocation list (CRL)
A list of certificates that have been revoked before their scheduled expiration date. Certificate revocation lists are maintained by the certificate authority and used, during a Secure Sockets Layer (SSL) handshake to ensure that the certificates involved have not been revoked.

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.

company application
An application that is designed for internal use inside a company.

Company Hub
An application that can distribute other specified applications to be installed on a mobile device. For example, Application Center is a Company Hub. See also Application Center.

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.

CRL
See certificate revocation list.

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.

DER encoded
Pertaining to a binary form of an ASCII PEM formatted certificate. See also Base64, PEM encoded.

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.

enterprise application
See company application.

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

in-house application
See company application.

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. 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.
2. A cryptographic mathematical value that is used to digitally sign, verify, encrypt, or decrypt a message. See also private key, public key.

key pair
In computer security, a public key and a private key. When the key pair is used for encryption, the sender uses the public key to encrypt the message, and the recipient uses the private key to decrypt the message. When the key pair is used for signing, the signer uses the private key to encrypt a representation of the message, and the recipient uses the public key to decrypt the representation of the message for signature verification.

L

library

1. A system object that serves as a directory to other objects. A library groups related objects, and allows users to find objects by name.
2. A collection of model elements, including business items, processes, tasks, resources, and organizations.

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 geolocational 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.

PEM encoded
Pertaining to a Base64 encoded certificate. See also Base64, DER encoded.

PKI See public key infrastructure.

PKI bridge
A Worklight Server concept that enables the User Certificate Authentication framework to communicate with a PKI.

poll To repeatedly request data from a server.

private key
In secure communication, an algorithmic pattern used to encrypt messages that only the corresponding public key can decrypt. The private key is also used to decrypt messages that were encrypted by the corresponding public key. The private key is kept on the user system and is protected by a password. See also key, public key.

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.

public key
In secure communication, an algorithmic pattern used to decrypt messages that were encrypted by the corresponding private key. A public key is also used to encrypt messages that can be decrypted only by the corresponding private key. Users broadcast their public keys to everyone with whom they must exchange encrypted messages. See also key, private key.

public key infrastructure (PKI)
A system of digital certificates, certification authorities, and other registration authorities that verify and authenticate the validity of each party involved in a network transaction. See also public key.

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 compnent 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 succesful 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.

X

X.509 certificate
A certificate that contains information that is defined by the X.509 standard.

Chapter 16. 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®, AIX, Cast Iron, Cognos, DataPower, DB2, developerWorks, Lotus, Passport Advantage, Power, PureApplication, Rational, Rational Team Concert, Redbooks, IBM SmartCloud, Tealeaf, Tivoli,, WebSphere, and Worklight are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at 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 17. 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

Chapter 18. Terms and conditions for information centers

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein. IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed. You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at www.ibm.com/legal/copytrade.shtml.

This information center is Built on Eclipse. (www.eclipse.org)

Index