IBM

IBM Worklight Foundation V6.2.0

Contents

Chapter 1. IBM Worklight Foundation V6.2.0 documentation

Welcome to the IBM® Worklight® Foundation documentation, where you can find information about how to install, maintain, and use the product.

Getting started

Overview of IBM Worklight Foundation

Product legal notices

What's New

System requirements for using IBM Worklight Foundation

IBM Worklight installation overview

Configuring Worklight Server

Tutorials and samples

Common tasks

Developing IBM Worklight applications

Testing with IBM Worklight Foundation

Deploying IBM Worklight projects

Administering IBM Worklight applications

Using the Application Center

Troubleshooting and support

Troubleshooting

Known limitations

Release notes

IBM Support home for IBM Worklight

IBM Worklight Foundation on Twitter

More information

Latest PDF file for this documentation

Mobile Application Developer skills for IBM Worklight Foundation

developerWorks blogs and articles

IBM Worklight Foundation blog

Mobile development community

IBM Redbooks

Chapter 2. Overview of IBM Worklight Foundation

Start here to learn about the IBM Worklight Foundation product.

With IBM Worklight Foundation, you can extend your business to mobile devices. IBM Worklight Foundation 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 Foundation 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 Foundation, see the following topics.

Introduction to mobile application development

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

IBM Worklight Foundation provides capabilities to help you respond to the fast-paced development of mobile devices. This flexible structure gives you more options when you implement your mobile communication channel, or release a new version of your application, be it a hybrid or native 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.

With IBM Worklight Foundation, you can develop mobile applications by using a spectrum of supported approaches. See Figure 1.
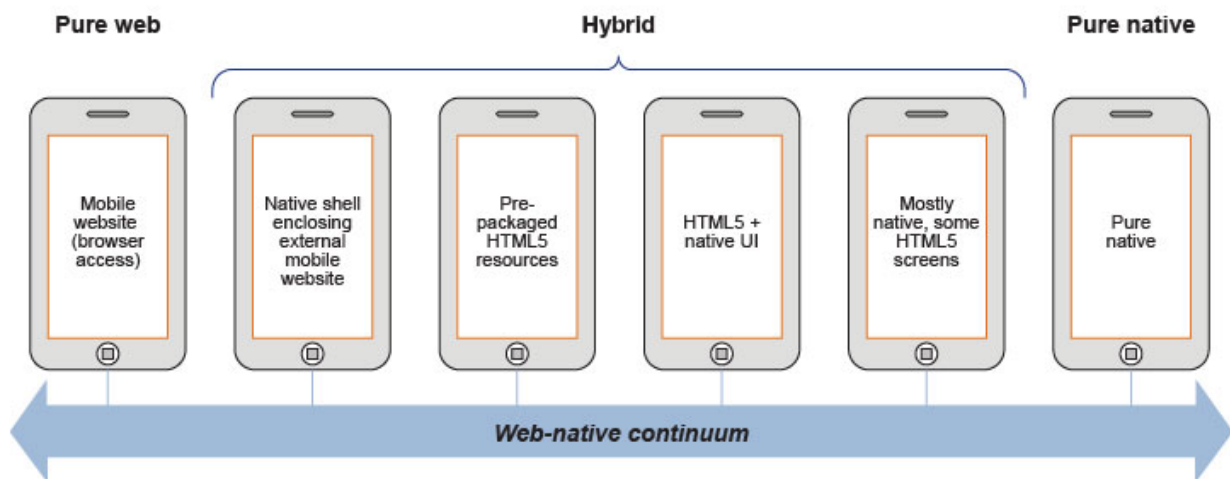
Figure 1. Spectrum of mobile app development approaches

Desktop and mobile website development

With the website development approach, users start their preferred browser and navigate to the enterprise website. The application runs inside the browser of the mobile device, and uses standard technologies such as HTML5, CSS3, and JavaScript to create the website. 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. On the downside, however, mobile web applications cannot access device functions such as contact list because they rely only on the browser and the associated web standards. In addition, if the website, and especially the enterprise back-end systems are not optimized for mobile interactions, user experience is less than pleasing. 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 and web development approaches with standards such as HTML5, JavaScript and CSS. Your hybrid application runs inside a native container and uses the browser engine to display the application interface. The interface is often based on HTML and JavaScript and can also incorporate native elements. The native container allows your application to access device capabilities that are not accessible to web applications, such as the accelerometer, camera, and local storage on a smartphone. These capabilities can be leveraged from JavaScript using Apache Cordova. The application is built using the mobile OS native IDE, such as Xcode. Similar to pure-native applications, hybrid applications are distributed through the application store of the platform.

As shown in the diagram, there are several possible approaches to hybrid development:

Native shell application enclosing an external mobile website
> Using this approach is similar to mobile website development, however your mobile website is displayed inside of a native shell that is provided by Worklight instead of the device browser. By using this approach, your mobile website is able to access device native functionality through APIs exposed by the native container. There are several drawbacks with this approach: drawing the app code and user interface remotely from a website might significantly downgrade user experience; when your only in-browser caching options are manual or through the HTML5 manifest, offline functioning is not optimized.

Pre-packaged HTML5 resources
> This is the most common approach when developing hybrid mobile applications. With this 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. Unlike the previous approach, the web resources are not loaded from an external website but are packaged within the application itself. In addition to leveraging the native functionality that is provided by a native container, the application can achieve better performance and responsiveness since all the resources required for application functionality are packaged within the app itself and do not need to be loaded over a network. By using this approach, your application acquires the ability to work and store data in offline

mode. To achieve the best user experience, it is important to optimize for various form factors by using techniques such as responsive web design.

Mixing web and native in code and UI (HTML5 + native UI/mostly native with some HTML5 screens)

Here, you mix web and native elements in one of two approaches: either you have a hybrid app to which you add native user interface components and gestures or you have a mostly native app and add HTML5 screens. By developing in this style, you are able, for example, to start your application with a native screen and move to a web screen at a later stage, or even mix native and web components on the same screen. There are several benefits:

- You fully leverage all platform functions such as accessing the camera or contact list from both native and JavaScript code.
- You achieve enhanced performance and user experience where you need it, by using native capabilities.
- You reuse code and web development skills by using HTML5/JavaScript/CSS where you can.

Pure native development

With the pure native development approach, you can create applications that are written for a specific platform and run on that platform only. Your applications achieve great performance and can fully leverage all platform functions such as accessing the camera or contact list, enabling gestures, or interacting with other applications on the device. To support platforms such as Android, iOS, Java™ ME, and Windows Phone, you must develop separate applications with different programming languages, such as Objective-C for iOS, or Java for Android, or C# for Windows Phone 8. Unlike desktop and mobile web applications, native and hybrid 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. In this table, you can find the different aspects of mobiles developments and the level of difficulty or possibilities that web development, hybrid development, hybrid mixed development, and native development, can bring for each aspect.

Aspect	Mobile website development	Native shell, external mobile website	Pre-packaged HTML5 resources	Mixing web and native in code and UI	Pure native development
Easy to learn	Easiest	Easiest	Medium	Harder	Hardest
Application performance	Slowest	Moderate	Good	Fastest	Fastest
Device knowledge required	None	Some	Some	Some	A lot

Table 1. Comparison of mobile development approaches (continued). In this table, you can find the different aspects of mobiles developments and the level of difficulty or possibilities that web development, hybrid development, hybrid mixed development, and native development, can bring for each aspect.

Aspect	Mobile website development	Native shell, external mobile website	Pre-packaged HTML5 resources	Mixing web and native in code and UI	Pure native development
Development lifecycle (build/test/deploy)	Shortest	Shortest	Medium	Medium	Longest
Application portability to other platforms	Highest	High	High	Medium	None
Support for native device functionality	Some	Most	Most	All	All
Distribution with built-in mechanisms	No	No	Yes	Yes	Yes
Ability to write extensions to device capabilities	No	No	Yes	Yes	Yes

Overview of IBM Worklight Foundation main capabilities

With IBM Worklight Foundation, 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 Foundation provides a framework that enables the development, optimization, integration, and management of secure mobile applications (apps). IBM Worklight Foundation 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 Foundation provides an SDK that includes libraries that you can access from native code.

Testing

IBM Worklight Foundation includes IBM Mobile Test Workbench for IBM Worklight Foundation for testing mobile applications. With the mobile testing capabilities of IBM Mobile Test Workbench for IBM Worklight Foundation, you can automate the creation, execution, and analysis of functional tests for IBM Worklight Foundation 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 Foundation 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 Foundation 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 Foundation 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 Foundation 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 Foundation 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 Foundation 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 Foundation 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 Foundation, you can define custom security handlers for any access to this flow of data. Because any access to data of a mobile application has to go through this server instance, you can define different security handlers for mobile applications, web applications, and back-end access. With this kind of granular security, you can define separate levels of authentication for different functions of your mobile application, or avoid sensitive information 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 Foundation 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 Foundation includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM Worklight Foundation applications and servers, and for monitoring server health.

Application publishing

IBM Worklight Foundation 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 Worklight components

IBM Worklight Foundation 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 Foundation, 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 Foundation, do not download your own Apache Cordova version.
* 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 Foundation application packages, and is in fact a collection of web applications, optionally packaged as an EAR (Enterprise Application ARchive) file that run on top of traditional application servers.

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

You can use Worklight Server for the following tasks:

* Empower hundreds of thousands of users with transactional capabilities and enable their direct access to back-end systems and cloud-based services.
* Configure, test, and deploy descriptive XML files to connect to various back-end systems by using standard Worklight Studio tools.

- Directly update deployed hybrid and web applications, without going through the different app stores (subject to the terms of service of the vendor).
- Automatically convert hierarchical data to JSON format for optimal delivery and consumption.
- Enhance 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 Foundation 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 Foundation 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

The Apache Cordova framework included with this version of IBM Worklight Foundation consists of the following components:

Platforms
- cordova-android: 3.4.0
- cordova-blackberry10: 3.4.0
- cordova-ios: 3.4.1
- cordova-windows: 3.4.0
- cordova-wp8: 3.4.0

Plugins
- org.apache.cordova.battery-status: 0.2.8
- org.apache.cordova.camera: 0.2.9
- org.apache.cordova.console: 0.2.8
- org.apache.cordova.contacts: 0.2.10
- org.apache.cordova.device-motion: 0.2.7
- org.apache.cordova.device-orientation: 0.3.6
- org.apache.cordova.device: 0.2.9
- org.apache.cordova.dialogs: 0.2.7
- org.apache.cordova.file: 1.1.0

- org.apache.cordova.file-transfer: 0.4.3
- org.apache.cordova.geolocation: 0.3.7
- org.apache.cordova.globalization: 0.2.7
- org.apache.cordova.inappbrowser: 0.4.0
- org.apache.cordova.media-capture: 0.3.0
- org.apache.cordova.media: 0.2.10
- org.apache.cordova.network-information: 0.2.8
- org.apache.cordova.splashscreen: 0.3.0
- org.apache.cordova.vibration: 0.3.8

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.

Worklight editions

IBM Worklight Foundation is available 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-warranted collection of self-contained development tools for IBM Worklight Foundation. It is available from the developerWorks® website. IBM Worklight Developer Edition contains the following components:

- Worklight Studio, which is available as an Eclipse plug-in.
- IBM Mobile Test Workbench for Worklight, which is available as an Eclipse plug-in. After you install Worklight Studio, you can then optionally install the test workbench in Eclipse.
- Command Line Interface for IBM Worklight Developers (CLI), which is available as an installable download. You can install and use CLI as an alternative to an integrated development environment (IDE) based Worklight Studio.

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®. IBM Worklight Consumer Edition and IBM Worklight Enterprise Edition contain the following components:

- The same three components as in IBM Worklight Developer Edition.
- A separate Worklight Server component, which is available as an IBM Installation Manager package

Choosing the appropriate edition

- Choose IBM Worklight Developer Edition if you want to start developing and learning about IBM Worklight Foundation. Worklight Studio and Command Line Interface for IBM Worklight Developers both include an embedded Worklight Development Server. This edition is useful to quickly develop and test applications.
- Choose IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition if you plan to develop and deploy production applications. These editions provide full production support. Customers can obtain support via the IBM Support home for IBM Worklight.

Note: Additionally, users of any edition can get help on the Stack Overflow community, where support is provided on a best-effort basis.

System requirements for using IBM Worklight Foundation

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

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

- The operating systems that support IBM Worklight Foundation, 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 application servers, database management systems, and other software that are required or supported by IBM Worklight Foundation

Matrix of features and platforms

IBM Worklight Foundation provides many features and supports many platforms.

The Mobile OS feature mapping for IBM Worklight technote on the IBM Support Portal lists the Worklight features that are available on each of the platforms that IBM Worklight Foundation supports.

Chapter 3. What's new

New features and changes extend your ability to work with IBM Worklight Foundation.

What's new in V6.2.0.1 interim fixes

Interim fixes provide patches and updates to correct problems and keep IBM Worklight Foundation current for new releases of mobile operating systems.

Interim fixes are cumulative. When you download the latest V6.2.0.1 interim fix, you get all of the fixes from earlier interim fixes.

Download and install the latest interim fix to obtain all of the fixes that are described in the following sections. If you install earlier fixes, you might not get all of the fixes described here.

New support for the latest iOS architectures

The latest interim fix for IBM Worklight Foundation V6.2.0.1 now provides support for arm64, in addition to support for armv7 and armv7s for hybrid app. You can build your new Worklight hybrid app with 32 bit (armv7 or armv7s) or with 64 bit (arm64), or for any combination of these architectures.

For more information about how to target those new architectures in your existing apps, see "Targeting new iOS architectures for your existing apps" on page 350.

iOS 9 support

If you use Xcode 7 to compile your apps install the latest interim fix and review the following sections to ensure that your apps continue to work on iOS 9.

Disabling bitcode-enabled Xcode builds

Starting with Xcode 7, bitcode is a default, but optional option for iOS apps. The bitcode option is not currently supported in IBM Worklight Foundation. To use the IBM Worklight Foundation SDK in any project that uses Xcode 7, you must disable bitcode.

Applications that are based on Apple watchOS 2 require the bitcode to be enabled and are currently not supported in IBM Worklight Foundation.

For more information, see "Disabling bitcode in Xcode builds" on page 621.

Support for dynamic `.tbd` libraries in Xcode 7

Xcode 7 replaces dynamic `.dylib` libraries with more lightweight `.tbd` files. Up to now, IBM Worklight Foundation projects link with `.dylib` libraries such as: `libc++.dylib`, `libstdc.dylib`, and `libz.dylib`. These libraries must be replaced with the corresponding `.tbd` libraries.

For guidelines, see the following topics:

15

- "Copying files of Native API applications for iOS" on page 618
- Adding Mobilefirst web capabilities to an existing native app

Enforcing TLS-secure connections in iOS apps

Apple's App Transport Security (ATS) is a new feature of iOS 9 that enforces best practices for connections between the app and the server. By default, this feature enforces some connection requirements that improve security. These include client-side HTTPS requests and server-side certificates and connection ciphers that conform to Transport Layer Security (TLS) version 1.2 using forward secrecy.

For **development purposes**, you can override the default behavior by specifying an exception in the `Info.plist` file in your app. However, in a full **production environment**, all iOS apps must enforce TLS-secure connections for them to work properly.

By applying the latest interim fix, the apps that you develop in IBM Worklight Foundation V6.0.0 and later automatically turn off transport security to allow all non-secure connections to the IBM Worklight Foundation development server.

Android 6.0 Marshmallow support

Android 6.0 Marshmallow introduces a number of changes that might require you to install the latest IBM Worklight Foundation interim fix for your app to work on Android 6.0.

Review the following sections to learn about changes in Android that might affect your apps and determine whether you need to install the latest interim fix for your app to support Android 6.0 Marshmallow.

Allowing runtime access to location services in Android 6.0 Marshmallow

In Android 6.0 Marshmallow and later, geo acquisition (location) permissions also require runtime permissions.

According to the new model, in addition to defining permissions at installation, users must allow or deny access to different features at runtime. Before an app accesses location services, it must check whether permission has already been granted and, if needed, request permission. Developers are responsible to perform the check before the app accesses any of the following methods in the WLDevice interface:
- startAcquisition
- acquireGeoPosition
- stopAcquisition

For more information, see For Android 6.0 Marshmallow and later.

Preventing Android 6.0 Marshmallow automatic backup of MobileFirst data

By default, app data on devices that are running under Android 6.0 Marshmallow is automatically backed up to Google Drive.

The IBM Worklight Foundation SDK holds app web resources, logs, and some other artifacts that are stored on the device. The volume of this data is potentially large and even if deleted can be easily recovered. Thus, there is no need for it to be backed up.

By applying latest interim fix, you can prevent most of the IBM Worklight Foundation SDK internal data from being backed up to Google Drive.

Note: After you apply the interim fix, some artifacts, for example residual logs might still be backed up to Google Drive.

What's new in IBM Worklight Foundation V6.2.0.1

IBM Worklight Foundation V6.2.0.1 fixes issues that were identified in V6.2.0.

For a complete list of issues that are fixed in V6.2.0.1, see Version 6.2.0 Fix Pack 1.

Documentation improvements

The V6.2.0.1 user documentation is updated to include the following improvements:

- Instructions on how to create an IBM Mobile Application Platform Pattern that uses a single DB2® virtual machine and uses the new database workload. See "Creating an IBM Mobile Application Platform Pattern that uses a single DB2 VM" on page 1228
- When you deploy a runtime environment that was built by using Worklight Studio V6.2.0 into Worklight Server V6.2.0.1, you can deploy applications and adapters of any version, from V5.0.6. Documentation to describe this feature was added in this service pack. For more information, see "Impact of migrating to a new version of Worklight Studio for applications already in production" on page 353.
- Instructions on how to deploy analytics to the cloud. See "Deployment of analytics to the cloud" on page 1241.
- The relevant cloud deployment topics highlight the fact that creating and configuring the reports database is optional.
- Instructions on how to implement adapter-based authenticators have been revised. See "Implementing adapter-based authenticators" on page 892.

Connection policy of HTTP adapters

In IBM Worklight Foundation V6.2.0, starting with Fix Pack 1, 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 743.

Upgrading from Worklight Server V6.2.0 to V6.2.0.x in a production environment

For more information about the specific installation instructions to upgrade to Worklight Server V6.2.0.1, see "Upgrading from Worklight Server V6.2.0 to V6.2.0.x in a production environment" on page 397.

New and updated tutorials and samples

New tutorials

New tutorials and their companion samples are delivered for authentication in native iOS and native Android applications:

- In Table 7: Adding native UI elements to hybrid applications (iOS and Android)
- In Table 9:
  - Form-based authentication
  - Adapter-based authentication
  - Custom authenticator and login modules
  - Using Worklight Server to authenticate external resources

Updated tutorials

Defects are fixed in the following tutorials. Their companion samples are unchanged.

- In Table 3: Worklight client-side development basics:
  - Optimizing your application for various environments
  - General_information for developing mobile web applications
  - General information for developing desktop applications
- Table 4: Worklight server-side development: Invoking adapter procedures from native Android applications

For more information about these new tutorials and samples, and how you can get started with IBM Worklight Foundation, see Chapter 4, "Tutorials and samples," on page 35.

What's new in IBM Worklight Foundation V6.2.0

IBM Worklight Foundation V6.2.0 includes new features and modifications. Some features of previous versions of the product are deprecated.

New product name

Starting with V6.2.0, IBM Worklight is renamed as IBM Worklight Foundation.

IBM expands the breadth of its MobileFirst application development platform capabilities and simplifies how you can consume these capabilities, by adding new components to the IBM Worklight platform:

- IBM Worklight Foundation (previously known as "IBM Worklight"): described in this documentation, this component provides mobile-optimized, standards-based middleware and tools for enterprise-grade mobile application and services creation, including development and runtime environments, an operation console and a private store.
- IBM Worklight Quality Assurance: this component enables you to analyze the user experience to help improve mobile applications, to collect beta test feedback and crash data, and to analyze user sentiment.

- IBM Worklight Application Scanning: this component helps you reduce security risks and lower support costs by identifying mobile application source code vulnerabilities early in the software development lifecycle so they can be fixed before deployment.

Note: This documentation is about IBM Worklight Foundation only. For more information about the other components and the IBM MobileFirst offering, see MobileFirst.

Flexible application development

Advanced native development and fast creation of hybrid and web applications.

Seamlessly mix web and native components in your application

In IBM Worklight Foundation V6.2.0, it is easier to mix web and native components in your application, allowing you to build a hybrid application with any composition of web and native capabilities. The following enhancements support this greater flexibility in hybrid app development.

- The architecture of hybrid applications (iOS and Android) now allows you to easily use native code when the application starts. You can add code at the beginning of the application lifecycle, before the Worklight framework is initialized, and have full control over when to initialize the Worklight framework, including being able to initialize it in the background. This flexibility allows you, for example, to add a custom splash screen or to control the default splash screen behavior, and to start an application with a native screen.
- Moreover, you can now navigate seamlessly between the native and web parts of your application without having to reauthenticate, and invoke both native and JavaScript WLClient APIs in any order. For example, the following scenarios are now possible:
  - Start the application in a web view page, connect to the Worklight server, and then login to access protected resources. You can then switch to a native view and access the same protected resources without the need to reauthenticate.
  - Start the application in a native page, connect to the Worklight server, and then login to access protected resources. You can then switch to a web view and access the same protected resources without the need to reauthenticate
- A new API has been introduced that allows you to easily send actions (events) and data between the native and web modules of your app. This makes it easier to build applications that mix hybrid and web components, for example:
  - You have an application with a JavaScript communication module and a native presentation layer. You can use the new API to easily send the data acquired by the communication module to the native layer, so that you can present it to the user in a native screen.
  - You can easily trigger complex native actions using JavaScript, without implementing a Cordova plugin. You can also call JavaScript code from native code.
  - You can easily show native and web components on the same screen, using this API for communication.

For more information, see "Developing hybrid applications" on page 453 and "Developing native applications" on page 614.

ARC now supported

WL 6.2 iOS SDK now supports ARC (Automatic Reference Counting). The default Worklight application that is generated for iPhone/iPad environment also supports ARC. Refer to the Apple documentation for more details on ARC.

Visibility of the splash screen at application startup

You can now choose to show or hide the splash screen image during application startup. For more information, see "Managing the splash screen" on page 472.

New Command Line Interface for IBM Worklight Developers

IBM Worklight Foundation provides command-line interface (CLI) tools as an alternative to the integrated development environment (IDE) of Worklight Studio.

Command Line Interface for IBM Worklight Developers provides basic scaffolding, project, and app support for native and hybrid development environments. This provision enables developers to use their preferred text editors to create mobile applications outside of the traditional Worklight Studio environment. When your apps are developed, the CLI provides the commands necessary to build and deploy them to a local development Worklight Server. Adapter creation, deployment, and local testing can be performed within the Command Line Interface for IBM Worklight Developers. These command-line tools can be used on their own, or in parallel with the Worklight Studio tools.

For more information about Command Line Interface for IBM Worklight Developers, see "Command Line Interface for IBM Worklight Developers" on page 640.

New capabilities in Worklight Studio for adding Worklight web capabilities to an existing native app

IBM Worklight Foundation provides the capability of adding a Worklight web view into an existing native Android, iOS, or Windows Phone 8 application by transforming it from pure native to a hybrid application. For more information, see "Adding Worklight web capabilities to an existing native app" on page 634.

New capabilities in IBM Worklight Application Framework for transferring data between your application and back-end services

When you create data objects from a service, you can now modify the default types of the data attributes that are created. You can also specify the format of the data that is transferred between the back-end service and your application when you modify the data type. For more information, see "Creation of data objects and usage of back-end services" on page 569.

Ability to troubleshoot common problems in the IBM Worklight Application Framework editor

With the new validation process that is available in the IBM Worklight Application Framework editor, you can now easily identify errors that might be contained in the views of your app. This validation process uses Eclipse infrastructure to report problems. For more information about how to troubleshoot IBM Worklight Application Framework views, see "Troubleshooting the views created with IBM

Worklight Application Framework" on page 585.

Note on Custom Challenge Handlers and Push API

Custom Challenge Handlers
Developers must create two custom challenge handlers, in native and JavaScript code, for each realm that is protected by a custom security test, and whose protected resources might be accessed from both native and web code.

Push For more information about the limitations on using the Push Notifications API, see "Using native and JavaScript push APIs in the same app" on page 858.

Extended user reach

A rich experience for a wide variety of mobile application users.

New capabilities in IBM Worklight Application Framework

* Capability to develop globalized applications: In IBM Worklight Foundation V6.2.0, every application that is created with IBM Worklight Application Framework can be globalized. All the view titles, field labels, and field hints come from a standard Dojo i18n message bundle. This NLS bundle is in the `common/nls` folder of your application, with a `messages.js` file. When the application starts, the run time loads the related locale-based bundle. For more information see, "Globalization of apps created with IBM Worklight Application Framework" on page 591.
* Push notifications enablement: You can now set up push notifications for your application in the **Application** tab of the IBM Worklight Application Framework editor. For more information, see "The **Application** tab of the IBM Worklight Application Framework editor" on page 566.
* New custom login option: You can now create a custom view, and define this new view as a login view instead of the default login view that is provided by IBM Worklight Application Framework at run time. For more information about how to configure a custom login view for your application, see "Setting up a custom login view" on page 586.

USSD Support

Unstructured Supplementary Service Data (USSD) is a communication technology that is used by GSM cellular telephones to send text messages between a mobile phone and an application program in the network. USSD support is now available in IBM Worklight Foundation V6.2.0. It works on both smart phones and feature phones.

USSD provides real-time interactive messaging and is particularly useful for building interactive menu driven applications. It provides a cost-effective alternative to mobile applications in emerging markets where feature phones are still widely used. No application installation is required on the client phones for USSD. Worklight now offers the following features:

* Accept incoming requests from a USSD gateway and map the USSD short codes to corresponding Worklight adapters
* Construct and respond with USSD menu options
* Invoke corresponding back-end services via Worklight adapters

For more information about USSD, see "USSD Support" on page 781.

Enforce language preference for Worklight messages

IBM Worklight Foundation supports several languages, and based on the language of the device, appropriate translations are used to display system messages. However, you can enforce restriction to display messages only in a certain set of languages. If your application supports only English and French, you can enforce a restriction so that Worklight displays the messages only in these languages.

For more information, see "Enforce language preference for Worklight messages" on page 955.

SSO between IBM Worklight Foundation and external services

Starting with IBM Worklight Foundation V6.2.0, you can use single sign-on (SSO) between IBM Worklight Foundation and external services by making use of the Worklight security framework to protect the external services. For more information, see "Using SSO between IBM Worklight Foundation and external services" on page 917.

Comprehensive mobile operations

Easily manage your portfolio of mobile assets.

Improved analytics

Analytics in IBM Worklight Foundation V6.2.0 is new and improved.

Starting with IBM Worklight Foundation V6.2.0, you can now easily use the IBM Worklight Analytics Platform by deploying the Analytics WAR file. For more information, see "Operational analytics" on page 1397.

Improved serviceability and log collection tools

Starting with IBM Worklight Foundation V6.2.0, you can now capture and receive uploaded client-side logs. For more information, see "Client-side log capture" on page 978.

Enhanced support for WSDL service discovery

- In the previous version of IBM Worklight, SOAP adapters that were generated with the services discovery wizard required the use of XML namespace prefixes for all the fields in the payload. In IBM Worklight Foundation V6.2.0, if the names of the elements are unique, the namespace prefixes do not need to be specified in the generated SOAP adapters. For more information about how to automatically generate adapters, see "Generating adapters with the services discovery wizard" on page 756.
- For WSDL-based services, a sample payload of each service is available from the Properties dialog box of each service in the Eclipse Project Explorer view. You can reuse this sample payload by copying it in other adapters that are created without IBM Worklight Application Framework.

Role-based access with Java Platform, Enterprise Edition (Java EE) security

Starting with IBM Worklight Foundation V6.2.0, the Worklight Server administration requires user authentication. The Worklight Server administration

has several Java Platform, Enterprise Edition (Java EE) security roles defined that you must map to the corresponding set of users.

For more information, see "Configuring user authentication for Worklight Server administration" on page 151.

Extended support for IBM Mobile Application Platform Pattern

Starting with IBM Worklight Foundation V6.2.0, support for IBM Mobile Application Platform Pattern extends to IBM PureApplication Service on SoftLayer. For more information, see "Deploying to the cloud" on page 1221.

Security

Enforce security at every level - user, device, data, and application.

JSONStore improvements

Starting with IBM Worklight Foundation V6.2.0, you can now easily use JSONStore for native apps. For more information, see "JSONStore overview" on page 787.

Simple data sharing

Starting with IBM Worklight Foundation V6.2.0, you can securely share lightweight information among a family of applications on a single device. For more information, see "Simple data sharing overview" on page 925.

Enhanced group support for the User Certificate Authentication feature for iOS and Android

Starting with IBM Worklight Foundation V6.2.0, you can use group support 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. For more information, see "User certificate authentication on the client" on page 1460.

Direct update improvements

Starting with IBM Worklight Foundation V6.2.0, direct update is improved in the following ways:

- Direct update is now part of the Worklight security framework. Apps developed in Worklight Studio V6.2.0 and later are defined as a security realm. Apps developed in earlier versions of Worklight Studio use the mechanism provided with V6.1.0. See "Direct Update as a security realm" on page 603.
- Support is extended to include Windows Phone 8. See "Direct updates of app versions to mobile devices" on page 599.
- Direct update requests can be cached on a CDN server. See "Serving direct update requests from a CDN" on page 605.
- Direct update process and interface can be customized. For information including typical scenarios, see "Customizing the direct update interface and process" on page 608.

Enhanced app authenticity for iOS and Android

Starting with IBM Worklight Foundation V6.2.0, app authenticity is enhanced for iOS and Android. For more information, see "Worklight application authenticity overview" on page 865.

Restricting database user permissions for Worklight Server runtime operations

When the databases are operational, you can create a database user with restricted privileges. You use this database user to perform database underlying operations from the Worklight Console. The user credentials appear in the application server configuration.

For more information, see "Restricting database user permissions for Worklight Server runtime operations" on page 75.

Integration with IBM Trusteer®

Starting with V6.2.0, IBM Worklight Foundation supports full integration with the IBM Trusteer Mobile SDK, for Android and iOS applications.

For more information, see "Integration with IBM Trusteer" on page 1477.

Configuring authenticators and realms

Documentation has been improved for the following topics:
- "Implementing basic authenticators" on page 881
- "Implementing form-based authenticators" on page 884

In V6.2.0, documentation has been added for "Implementing custom authenticators" on page 887.

New package for IBM Worklight Developer Edition

IBM Worklight Developer Edition V6.2.0 is reorganized into a collection of self-contained, easy-to-install development tools for IBM Worklight Foundation.

New components

Before IBM Worklight Foundation V6.2.0, two separate Worklight plug-ins were available from developerWorks:
- The Worklight Studio plug-in, which was known as IBM Worklight Developer Edition.
- The IBM Mobile Test Workbench for Worklight plug-in, which could be installed on top of IBM Worklight Developer Edition.

With the introduction of the Command Line Interface for IBM Worklight Developers and a common Worklight Studio in all Worklight editions (see following section), IBM Worklight Developer Edition now provides more than the single Worklight Studio plug-in that was delivered in earlier versions. With IBM Worklight Foundation V6.2.0, IBM Worklight Developer Edition consists of three separately installable components:
- Worklight Studio that now includes an authenticity feature
- IBM Mobile Test Workbench for Worklight

- Command Line Interface for IBM Worklight Developers

IBM Worklight Developer Edition packages stand-alone development components and does not require the installation of database or of application servers, whereas IBM Worklight Consumer Edition and IBM Worklight Enterprise Edition both provide development environment and server components for production. For more information, see "Worklight editions" on page 12.

Common Worklight Studio for all Worklight editions

In IBM Worklight Foundation V6.2.0, the instances of Worklight Studio that are delivered in IBM Worklight Developer Edition, IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition are now the same:

- Worklight Studio that is installed from the IBM Worklight Developer Edition now includes the authenticity feature, which was previously only available in the IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition. Developers can now create production-ready applications with any edition of Worklight Studio.
- The internal Worklight Development Server that is embedded in Worklight Studio does not include the authenticity feature. Deploying and testing the authenticity feature requires IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition. You must install Worklight Server, configure it in Worklight Studio as an external server, and deploy the application to Worklight Server.

New Worklight Studio startup behavior

The startup behavior of IBM Worklight Studio is changed to improve Worklight Development Server startup time.

In the previous release of IBM Worklight, every project has an associated WAR file and you deploy it to the Worklight Development Server during standard development activities. These WAR applications stay deployed to the server unless you remove the unnecessary ones from the server. When Worklight Studio restarts, you might start Worklight Development Server to continue your development. Depending on the number of deployed projects, a Timeout error with the Worklight Development Server might occur because of the 60 seconds limit imposed by Eclipse settings.

From IBM Worklight Foundation V6.2.0, the server startup time is improved. Every project still has an associated WAR file and you deploy it to Worklight Development Server during the normal Worklight Studio usage workflow. However, Worklight Studio now remembers the last deployed project to the server. When Worklight Studio restarts, all deployed Worklight WAR applications are deleted from the server, except from the last one, improving the server startup time.

Note: You can inhibit this new behavior and make it work exactly as before by using the following steps:

- Close Worklight Studio.
- Go to the following path: /<workspace_path>/.metadata/.plugins/ org.eclipse.core.runtime/.settings
- Open com.worklight.studio.plugin.prefs file with a text editor.
- Add **com.worklight.studio.plugin.avoidServerCleanup=true**.
- Save the file and restart Worklight Studio.

Improvements in the installation and upgrade of Worklight Server V6.2.0

IBM Worklight Foundation V6.2.0 includes a new architecture of the Worklight Server components and improvements in the installation and upgrade process.

New architecture of the Worklight Server components

Starting with IBM Worklight Foundation V6.2.0, there is a new architecture for Worklight Server, which is composed of the following components:

- The Worklight runtime environments, which are packaged as web applications (WAR file) and host one or more Worklight applications.
- The Worklight Console application, which is the web-based interface that is used to run administration tasks on the mobile application. The Worklight Console is supported by the Administration Services application, and can manage several Worklight runtime environments.
- The Administration Services application, which host all the REST services and administration tasks.
- The Application Center
- The Worklight Operational Analytics

The Worklight Console and Administration Services constitute the Worklight Server administration. For more information about these components, see "Introduction to the Worklight Server components" on page 73.

New installation procedure for the Worklight Server

Due to the new architecture of the Worklight Server, new steps are added to the installation of the Worklight Server. You must install the Worklight Console and Administration Services applications, and create an administration database for these components. For more information, see the installation procedure in "Installing Worklight Server" on page 70.

Installing a server farm

Starting with IBM Worklight Foundation V6.2.0, Worklight Server provides a specific plug-in so that instances of application servers can become server farm nodes. For more information, see "Installing a server farm" on page 164.

Application server configuration with Java Management Extensions (JMX)

In IBM Worklight Foundation V6.2.0, you must configure your application server to enable Java Management Extensions (JMX) communication between the Administration Services and the Worklight runtime environment. The administration tasks are run through standard JMX (Java Management Extension) calls. Starting with IBM Worklight Foundation V6.2.0, the Worklight runtime environment registers an MBean called `com.worklight.core.jmx.ProjectManagementMXBean` and exposes the JMX API with this MBean.

For more information, see "Configuration of the application server" on page 107.

Applying fix packs and interim fixes without interruption

With IBM Worklight Foundation V6.2.0, you can perform a rolling upgrade to apply a fix pack or an interim fix to an installation of Worklight Server V6.2.0 without downtime of the Worklight runtime environment. Performing this rolling upgrade ensures that there is no interruption of service for the mobile applications that query the Worklight Server.

This procedure applies to a Worklight Server installation, including the Administration Services, the Worklight Console, and the Worklight runtime environment, but does not apply to the Application Center.

For more information, see "Rolling upgrade procedure to apply a fix pack to IBM Worklight Foundation V6.2.0" on page 398.

Migrating to a new version of Worklight Studio for applications already in production

When you deploy a runtime environment that was built using Worklight Studio V6.2.0 into Worklight Server V6.2.0.1, you can deploy applications and adapters of any version, from V5.0.6. For more information, see "Impact of migrating to a new version of Worklight Studio for applications already in production" on page 353.

Improvements in administration tools

IBM Worklight Foundation V6.2.0 includes improvements to Worklight Console and new ways to administer Worklight projects.

Worklight Console improvements

- Since V6.2.0, Worklight Console is a stand-alone web application. In earlier versions, it is embedded in a project WAR file, which requires a separate URL to access a separate console for each project.
- One Worklight Console can now manage several Worklight runtime environments that are deployed on the same topology (either a WebSphere® Application Server Network Deployment cluster, a single server, or a server farm). A new home page in the console lists all the runtime environments. You click the name of the runtime environment to display the details of the applications and adapters that you want to manage.
- Each Worklight Console is protected by a standard Java Platform, Enterprise Edition security configuration that requires you to log on with one of the configured roles. The features available to you in Worklight Console depend on your designated role.
- A status bar indicates any administration problems that might occur, and provides a link to further details that are displayed in a new activity log.

For more information about Worklight Console improvements, see "Administering Worklight applications with Worklight Console" on page 1248.

New administration tools

IBM Worklight Foundation V6.2.0 includes the following new ways to administer Worklight projects: the `wladm` command, `<wladm>` Ant tasks, and REST services.

- Public REST services support all administration operations, including listing, deploying, deleting, and changing applications and adapters. See "REST Services API" on page 1016.

- The `<wladm>` Ant tasks support the same features as the REST services. The Ant tasks are defined in the `worklight-ant-deployer.jar` file. They support SSL and password encryption. See "Administering Worklight applications through ANT" on page 1258.
- The `wladm` command provides a command-line version of the `<wladm>` Ant tasks. The `wladm` command is provided as a set of scripts. See "Administering Worklight applications through the command line" on page 1275.

Push notification enhancements

In addition to event-source based notification, you can now enable tag-based notification and broadcast notification. You might have to run migration scripts.

In addition to push notification as based on user subscription to an event source, IBM Worklight Foundation V6.2.0 implements an infrastructure with improved scalability, which brings the following features:

- Tag-based notification: In the descriptor file of the Worklight application, you add tags that define topics of interest, such as customer categories, and the application can subscribe to those tags to receive notification. For more information, see "Setting up Tag-based notifications" on page 849.
- Broadcast notification: Any push-enabled application is also enabled for broadcast. For more information, see "Broadcast notifications" on page 847. To send a broadcast notification, use the `WL.Server.sendMessage` method. For more information about sending a notification message, see "Sending push notifications" on page 852.
- New JavaScript and REST APIs to manage device subscription: For more information, see the documentation of the `subscribeTag` and `unsubscribeTag` methods of the WL.Client.Push class.

Augmented support of mobile environments and operating systems

IBM Worklight Foundation V6.2.0 offers extended capabilities in terms of supported mobile environments and operating systems.

Updated operating systems, hardware, and software support

IBM Worklight Foundation V6.2.0 includes support for recent levels of mobile environments and operating systems.

For more information, see "System requirements for using IBM Worklight Foundation" on page 13.

New support for the latest iOS architectures

Armv7s (32 bit) and arm64 (64 bit) are new versions of ARM Instruction Set Architecture. IPhone 5, iPhone 5C, and fourth-generation iPad devices are powered by armv7s processors. IPhone 5S, iPad Air, and second-generation iPad mini (with Retina display) devices are powered by arm64 processors.

You can use Xcode 4.5 and later to build your app with support for both armv7 and armv7s. You can use Xcode 5.0.1 and later to build your app with support for armv7, armv7s, and arm64.

- For hybrid apps, IBM Worklight Foundation V6.2.0 now provides support for armv7s, in addition to support for armv7. You can build your new Worklight hybrid app for armv7 or for armv7s, or for both.
- For the native Worklight API, IBM Worklight Foundation V6.2.0 now provides support for armv7s and arm64, in addition to armv7. You can build your new Worklight native app with 32 bit (armv7 or armv7s) or with 64 bit (arm64), or for any combination of these architectures.

Note: The combined binary that includes a slice for armv7, armv7s, and arm64 requires a minimum deployment target of iOS 5.1.1 or later. If you want to support earlier iOS versions, back to iOS 5.0, you must change the selection in the Xcode **Architectures** menu to exclude arm64.
To target armv7s or arm64 in Xcode, you must make sure that all the libraries that you use and that are external to IBM Worklight Foundation also support the architectures that you want to target.

For more information about how to target those new architectures in your existing apps, see "Targeting new iOS architectures for your existing apps" on page 350.

Third-party libraries

IBM Worklight Foundation V6.2.0 is now based on Cordova 3.4.1. The upgrade process for Cordova configuration is automated.

For more information about the libraries supported by IBM Worklight Foundation V6.2.0, see "System requirements for using IBM Worklight Foundation" on page 13.

Removed environments

With IBM Worklight Foundation V6.2.0, 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.

Change from Tealeaf® V8 to Tealeaf V9
- Tealeaf Client SDK V9 is delivered under the optional feature **Tealeaf Client SDK** in Worklight Studio V6.2.0. In previous version of IBM Worklight Foundation, the name of this optional feature was Analytics.
- `WL.Analytics` JavaScript API no longer delegates to Tealeaf TLT JavaScript API. `WL.Analytics` JavaScript API now delegates to Worklight internal SDK and achieves the same results as the previous behavior.
- For Worklight V6.1 projects in which the Analytics optional feature is already enabled and that you upgrade to Worklight Studio V6.2.0: the Tealeaf artifacts are replaced, including `uicandroid.jar` and `TLFConfigurableItems.properties` on Android, all Tealeaf `.h`, `.plist` and `libTLFLib.a` files on iOS, and `tealeaf.min.js`.

  Note: The three Tealeaf import statements are commented out in the `main.m` file of iPhone and iPad environments. This is necessary to avoid compile errors that are related to unresolved import statements, as the Tealeaf header files were removed from the Worklight SDK. The header files are available as part of the Tealeaf Client SDK optional feature. Applications that use the Tealeaf Client SDK optional feature must add the appropriate header file imports based on the Tealeaf version that is used.

- Applications are not automatically instrumented with Tealeaf Client SDK API calls. Make sure to instrument the application appropriately to capture the required events for Tealeaf CX server replay and analysis.

Improved Worklight API

IBM Worklight Foundation V6.2.0 improves and extends the APIs that you can use to develop mobile applications.

New C# client-side API for Windows Phone 8

IBM Worklight Foundation V6.2.0 includes new C# client-side API for Windows Phone 8 that you can use to develop apps for Windows Phone 8.

For more information, see "C# client-side API for Windows Phone 8 apps" on page 1006.

Updated JavaScript client-side API

IBM Worklight Foundation V6.2.0 includes updates in its JavaScript client-side API, as follows:
- New API:
  - `WLApp.sendActionToNative`:

    With this API, you can send any number of actions and optional data objects from JavaScript code to native code.
  - `WLApp.addActionReceiver`:

    With this API, you can add an action receiver.
  - `WLApp.removeActionReceiver`:

    With this API, you can remove an action receiver.

  For more information, see WLApp
- Deprecated API:
  - `WL.Client.init - connectOnStartup`:

    Instead, your app must call WL.Client.connect method, as defined in the WL.Client class, only when it requires to connect to Worklight Server.

Updated JavaScript server-side API

IBM Worklight Foundation V6.2.0 includes updates in its JavaScript server-side API that you can use to extend the Worklight Server, as follows:
- New API:
  - `WL.Server.createUSSDEventHandler(aFilter, aHandler)`:

    This API creates an USSD event handler. For more information, see WL.Server.
  - `WL.Server.createUSSDResponse(content, contentType, isLastMsg)`:

    This API creates the USSD response.. For more information, see WL.Server.

For more information, see "JavaScript server-side API" on page 1007.

Updated Objective-C client-side API for iOS

IBM Worklight Foundation V6.2.0 includes updates in its Objective-C client-side API to develop native apps on iOS, as follows:

- New API:
  - `WLClient.login` and `WLClient.logout`:

    This API allows you to log in to or log out of specific security realms, so triggering authentication defined by these realms. The HTTP context (for example, cookies and headers) is shared by both the native and JavaScript `WL.Client`. As a result, after the user authenticates by using native `WL.Client`, the authentication state will also exist in the JavaScript `WL.Client` and the other way around. Users will not have to re-authenticate when they transition between native and web views.

    Note: The equivalent JavaScript API also exists.

    For more information, see WLClient.
  - `WLClient.wlConnectWithDelegate`:

    This API initializes the communication with the Worklight Server by using the connection properties and application ID taken from the `worklight.plist` file. At this point, the server also checks the validity of the application version. This method must be called before any other WLClient method. When the server returns a successful response, the delegate's onSuccess method is called. If an error occurs, the delegate's onFailure method is called. This method accepts a **timeout** key in its options parameter.

    For more information, see WLClient.
  - `WL.showSplashScreen` and `WL.hideSplashScreen`:

    These methods will show or hide a splash screen. A root view controller must be defined before calling this method.

    For more information, see WL.
  - `WL.sendActionToJS(action, data)`:

    With this API, you can send any number of actions and optional data objects from native Objective-C code to JavaScript code.

    For more information, see WL.
  - `WLActionReceiver`:

    A protocol that a class must implement to receive actions that JavaScript code sends to native code with the `sendActionToNative` API.

    For more information, see WLActionReceiver.
  - `WL.addActionReceiver(<WLActionReceiver>)`:

    With this API, you can add a new action receiver.

    For more information, see WL.
  - `WL.removeActionReceiver(<WLActionReceiver>)`:

    With this API, you can remove an action receiver.

    For more information, see WL.

For more information, see "Objective-C client-side API for iOS apps" on page 1005.

Updated Java client-side API for Android

IBM Worklight Foundation V6.2.0 includes updates in its Java client-side API to develop native apps on Android, as follows:
- New API:
  - `WLClient.login` and `WLClient.logout`:

    This new API allows you to log in to or log out of specific security realms, so triggering authentication defined by these realms. The HTTP context (for

example, cookies and headers) is shared by both the native and JavaScript `WL.Client`. As a result, once the user authenticates using native `WL.Client`, the authentication state will also exist in the JavaScript `WL.Client` and the other way around. This means that users will not have to re-authenticate when they transition between native and web views.

> **Note:** The equivalent JavaScript API also exists.

> For more information, see WLClient.

– `WLClient.connect`:

This new API sends an initialization request to the Worklight Server, establishes a connection with the server, and validates the application version.

For more information, see WLClient.

– `WL.sendActionToJS:action withData:data`:

With this API, you can send any number of actions and optional data objects from native Java code to JavaScript code.

For more information, see WL.

– `WLActionReceiver`:

An interface that a class must implement in order to be able to receive actions that JavaScript code sends to native code with the `sendActionToNative` API.

For more information, see WLActionReceiver.

– `WL.addActionReceiver (WLActionReceiver)`:

With this API, you can add a new action receiver.

For more information, see WL.

– `WL.removeActionReceiver (WLActionReceiver)`:

With this API, you can remove an action receiver.

For more information, see WL.

- Deprecated API:
  – `WLDroidGap`:

This class has been deprecated and should not be used in newer hybrid applications.

For more information, see "Java client-side API for Android apps" on page 1006.

Miscellaneous modifications

IBM Worklight Foundation V6.2.0 includes other miscellaneous modifications.

Changes in Worklight settings page

When you create a new Android or iOS application with IBM Worklight Foundation V6.2.0, the Worklight settings page is now disabled by default in the `application-descriptor.xml` file

For more information, see "Settings page to change the server URL" on page 507.

Changed location of Ant builder file

Since IBM Worklight Foundation V6.2.0, `worklight-ant-builder.jar` file is now included in the Command Line Interface for IBM Worklight Developers. By default, the `worklight-ant-builder.jar` file is installed in the following location: `<CLI Install Path>`/public/worklight-ant-builder.jar. For example, on OSX, the default CLI Install Path is /Applications/IBM/Worklight-CLI. If you use the default

installation path, the Ant task is installed here: `/Applications/IBM/Worklight-CLI/public/worklight-ant-builder.jar`. For more information, see "Ant tasks for building and deploying applications and adapters" on page 1190.

Upgraded version of Cordova

IBM Worklight Foundation V6.2.0.2 uses an upgraded version of Cordova. For more information about the version of Cordova that is used, see Software Product Compatibility Reports, click the **Supported Software** tab, and then click **Runtime Environment**.

Chapter 4. Tutorials and samples

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

You can get started with IBM Worklight Foundation and learn how to develop mobile applications with IBM Worklight Foundation by studying the following tutorials and samples:
- "Tutorials"
- "Samples" on page 45

Before you start developing mobile applications with IBM Worklight Foundation, consider reviewing the best practices on the developerWorks Best practices page 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 45.

Important: These materials are designed 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 45, and may include applicable third-party licenses. Review the third-party licenses before you use 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 Foundation.

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 Foundation.
- The sample, if any, is a compressed .zip file that provides pieces of code or script files to support the module. If a module has some exercises, a companion sample provides the solutions to these exercises.

The modules and companion samples of the tutorials are organized in categories:
- Setting up your development environment: These tutorials describe how to set up your development environment to work with IBM Worklight Foundation.
- Hello Worklight: These tutorials describe how to create your first Worklight app and preview it in different mobile operating systems.
- Worklight client-side development basics: These tutorials describe how to use basic Worklight APIs to develop your apps, build a multi-page application, work with the user interface framework, and debug and optimize your apps. Before work in each specific environment, make sure that you know the general information that is also provided.

- Worklight server-side development: These tutorials describe how to develop the server code (adapters) that your mobile application requires to integrate with enterprise back-end applications and cloud services.
- Advanced client-side development: These tutorials 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.
- Working offline: These tutorials describe how to create applications that can run offline.
- Adding native functionality to hybrid apps with Apache Cordova: These tutorials describe how to use Apache Cordova with IBM Worklight Foundation, and how to use native pages in hybrid applications.
- Notifications: These tutorials describe how to send notifications to your mobile applications with IBM Worklight Foundation.
- Authentication and security: These tutorials describe how to protect your applications and adapter procedures against unauthorized access by using authentication, login modules, and device provisioning.
- Advanced topics: These tutorials describe advanced topics that you can use with IBM Worklight Foundation, such as how to develop by using shells, or how to handle notifications.
- Moving to production: These tutorials describe how to move the apps that you create from your development environment to the production environment.

Note: Compared to the previous version of IBM Worklight Foundation, 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 tables.

Table 2. Setting up your development environment

Module	Sample (if any)	Description
Setting up your Worklight development environment		This module explains how to set up your environment to work with IBM Worklight Foundation.
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.
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.

Table 2. Setting up your development environment (continued)

Module	Sample (if any)	Description
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.

Table 3. Hello Worklight

Module	Sample (if any)	Description
Creating your first hybrid Worklight application	Worklight project	This module explains how to create your first hybrid mobile application with IBM Worklight Foundation.
NEW in 6.2.0: Creating your first native iOS Worklight application	Worklight project and associated Native project	This module explains how to create your first native iOS mobile application.
NEW in 6.2.0: Creating your first native Android Worklight application	Worklight project and associated Native project	This module explains how to create your first native Android mobile application. **Note:** This module has the same Worklight project as the module *Creating your first native iOS Worklight application.*
NEW in 6.2.0: Creating your first native Windows Phone 8 Worklight application	Worklight project and associated Native project	This module explains how to create your first native Windows Phone 8 mobile application. **Note:** This module has the same Worklight project as the module *Creating your first native iOS Worklight 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.
Previewing your application in Windows 8		This module explains how to preview your application in the Windows Phone 8 environment.
Previewing your application on Mobile Web and Desktop Browser		This module explains how to preview your application in the mobile web environment.

Table 4. Worklight client-side development basics

Module	Sample (if any)	Description
Learning Worklight hybrid client-side API	Worklight project	This module explains the basics of the Worklight hybrid client-side API.
Building a multi-page application	Worklight project	This module explains how to build a multipage application with IBM Worklight Foundation.
Working with UI frameworks		This module explains how to work with the user interface (UI) frameworks of IBM Worklight Foundation.
Debugging your applications		This module explains how to debug the client applications.
Optimizing your application for various environments		This module explains how to optimize the application code for specific environments.
General information when developing for iOS		This module gives some general information that you must know to develop apps for the iOS environment.
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.

Table 5. Worklight server-side development

Module	Sample (if any)	Description
Adapter framework overview		This module gives general information about adapters and the adapter framework in IBM Worklight Foundation, and how to work with adapters.
HTTP adapter - Communicating with HTTP back-end systems	Worklight project	This module explains how to work with adapters to communicate with HTTP back-end systems.

Table 5. Worklight server-side development (continued)

Module	Sample (if any)	Description
SQL adapter - Communicating with SQL database	Worklight project	This module explains how to work with adapters to communicate with SQL databases. **Note:** This module has the same Worklight project 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		This module explains how to work with adapters to communicate by using Java Message Service (JMS).
Invoking adapter procedures from hybrid client applications	Worklight project	This module explains how to call the adapter procedures from hybrid client applications.
NEW in 6.2.0: Invoking adapter procedures from native iOS applications	Worklight project and associated Native project	This module explains how to call the adapter procedures from native iOS applications.
NEW in 6.2.0: Invoking adapter procedures from native Android applications	Worklight project and associated Native project	This module explains how to call the adapter procedures from native Android applications. **Note:** This module has the same Worklight project as the module *Invoking adapter procedures from native iOS applications*.
NEW in 6.2.0: Invoking adapter procedures from native Windows Phone 8 applications	Worklight project and associated Native project	This module explains how to call the adapter procedures from native Windows Phone 8 applications. **Note:** This module has the same Worklight project as the module *Invoking adapter procedures from native iOS applications*.
Invoking adapter procedures from native Java Platform, Micro Editions (Java ME) applications	Worklight project and associated Native project	This module explains how to use Java API to develop Java Platform, Micro Edition (Java ME) applications. **Note:** This module has the same Worklight project as the module *Invoking adapter procedures from native iOS applications*.
Advanced adapter usage and mashup	Worklight project	This module explains advanced details on how to use adapters.
Using Java in adapters	Worklight project	This module explains how to use Java in adapters.

Table 6. Advanced client-side development

Module	Sample (if any)	Description
Overview of client technologies		This module explains the technologies that support Worklight clients.

Table 6. Advanced client-side development (continued)

Module	Sample (if any)	Description
Common UI controls	Worklight project	This module explains the common user-interface controls in IBM Worklight Foundation.
Supporting multiple form-factors using Worklight skins		This module explains how you can support multiple form factors by working with skins in IBM Worklight Foundation.
Enabling translation	Worklight project	This module explains how to enable translation of the client applications.
Using Direct Update to quickly update your application	Worklight project	This module explains how to automatically update your applications with new versions of their web resources.
NEW in 6.2.0: Using CLI to create, build, and manage Worklight project artifacts		This module explains how you can use the Command Line Interface for IBM Worklight Developers to create, build, and manage your Worklight projects.
NEW in 6.2.0: Remote controlled client-side log collection		This module explains how you can remotely control the collection of the client-side logs of your application.

Table 7. Working offline

Module	Sample (if any)	Description
Working offline	Worklight project	This module explains how to detect application connectivity failures, and what actions are available to deal with the failures.
Storing sensitive data in encrypted cache	Worklight project	This module explains how to work with the encrypted cache of the mobile device.
NEW in 6.2.0: JSONStore - JavaScript API	Worklight project	This module introduces JSONStore, and how you can work with JSON documents in hybrid apps.
NEW in 6.2.0: JSONStore - Objective-C API	Worklight project and associated Native project	This module complements the module "JSONStore - JavaScript API" with further information about JSONStore Objective-C API for iOS. **Note:** This module has the same Worklight project as the module *JSONStore - JavaScript API.*
NEW in 6.2.0: JSONStore - Java API	Worklight project and associated Native project	This module complements the module "JSONStore - JavaScript API" with further information about JSONStore Java API for Android. **Note:** This module has the same Worklight project as the module *JSONStore - JavaScript API.*

Table 8. Adding native functionality to hybrid applications with Apache Cordova

Module	Sample (if any)	Description
Apache Cordova overview		This module explains what Apache Cordova is, and how to use it with IBM Worklight Foundation.
iOS - Using native pages in hybrid applications	Worklight project	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	Worklight project	This module explains how to use Apache Cordova plugs-in to add native functionality to hybrid applications that are developed for the iOS environment.
Android - Using native pages in hybrid applications	Worklight project	This module explains how to use native pages in hybrid applications that are developed for the Android environment. **Note:** This module has the same Worklight project as the module *iOS - Using native pages in hybrid applications*.
Android - Adding native functionality to hybrid application with Apache Cordova plugin	Worklight project	This module explains how to use Apache Cordova plug-ins to add native functionality to hybrid applications that are developed for the Android environment.
Windows Phone 8 - Using native pages in hybrid applications	Worklight project	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 Worklight project as the module *iOS - Using native pages in hybrid applications*.
Windows Phone 8 - Adding native functionality to hybrid application with Apache Cordova plugin	Worklight project	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.
NEW in 6.2.0.1: iOS_Adding_native_UI_elements_to_hybrid_applications	Worklight project	This module explains how to add native UI elements to hybrid applications that are developed for the iOS environment.
NEW in 6.2.0.1: Android_-_Adding_native_UI_elements_to_hybrid_applications	Worklight project	This module explains how to add native UI elements to hybrid applications that are developed for the Android environment.

Table 9. Notifications

Module	Sample (if any)	Description
Push notifications in hybrid applications	Worklight project	This module explains how to configure mobile devices to receive messages that are pushed from a server.
Push notifications in native iOS applications	Worklight project and associated Native project	This module explains how to use Worklight API to manage push notifications in native iOS applications.

Table 9. Notifications (continued)

Module	Sample (if any)	Description
Push notification in native Android applications	Worklight project and associated Native project	This module explains how to use Worklight API to manage push notifications in native Android applications. **Note:** This module has the same Worklight project as the module *Push notifications in native iOS applications*.
SMS notifications	Worklight project	This module explains how to configure mobile devices to receive notifications through SMS messages that are pushed from a server.
Two-way SMS communication	Worklight project	This module explains how to communicate with Worklight Server from any mobile device by using the SMS channel.

Table 10. Authentication and security

Module	Sample (if any)	Description
Authentication concepts		This module explains how to protect your applications and adapter procedures against unauthorized access by using authentication.
Form-based authentication in hybrid applications	Worklight project	This module explains how to work in hybrid applications with form-based authentication.
NEW in 6.2.0.1: Form-based authentication in native iOS applications	Worklight project and associated Native project	This module explains how to work in native iOS applications with form-based authentication
NEW in 6.2.0.1: Form-based authentication in native Android applications	Worklight project and associated Native project	This module explains how to work in native Android applications with form-based authentication
Adapter-based authentication in hybrid applications	Worklight project	This module explains how to work in hybrid applications with the adapter-based authentication.
NEW in 6.2.0.1: Adapter-based authentication in native iOS applications	Worklight project and associated Native project	This module explains how to work in native iOS applications with adapter-based authentication
NEW in 6.2.0.1: Adapter-based authentication in native Android applications	Worklight project and associated Native project	This module explains how to work in native Android applications with adapter-based authentication
Custom Authenticator and Login Module in hybrid applications	Worklight project	This module explains how to work in hybrid applications with custom login modules and authenticators when the default ones do not suffice.

Table 10. Authentication and security (continued)

Module	Sample (if any)	Description
NEW in 6.2.0.1: Custom Authenticator and Login Module in native iOS applications	Worklight project and associated Native project	This module explains how to work in native iOS applications with custom login modules and authenticators when the default ones do not suffice.
NEW in 6.2.0.1: Custom Authenticator and Login Module in native Android applications	Worklight project and associated Worklight project	This module explains how to work in native Android applications with custom login modules and authenticators when the default ones do not suffice.
Using LDAP Login Module to authenticate users with LDAP server in hybrid applications	Worklight project	This module explains how to work in hybrid applications 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	Worklight project	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.
Client X.509 Certificate Authentication and User Enrollment	Worklight project	This module explains how to work with the User Certificate Authentication feature by using the embedded PKI.
NEW in 6.2.0: Using the Worklight Server to authenticate external resources	Worklight project	This module explains how you can use the Worklight Server to authenticate external resources.

Table 11. Advanced topics

Module	Sample (if any)	Description
Shell development concepts	Worklight project	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.
Location services in Worklight applications	Worklight project, Native iOS project, and Native Android project	This module explains how to use geo-location services with IBM Worklight Foundation in hybrid applications.

Table 11. Advanced topics (continued)

Module	Sample (if any)	Description
Integrating server-generated pages in hybrid applications	Worklight project	This module explains how to remotely load dynamic content, where the code (HTML, CSS, and JavaScript) is hosted externally.
Using Worklight application as a container for server-generated pages	Worklight project	This module explains how to remotely load dynamic content, where the code (HTML, CSS, and JavaScript) is hosted externally.
Container for advanced pages	Worklight project	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.
Accelerating application development by reusing resources		This module explains how you can use components and project templates with IBM Worklight Foundation.
Creating an application with IBM Worklight Application Framework	Worklight project	This module explains how to create an application with IBM Worklight Application Framework.
Testing Worklight mobile applications with the Mobile Test Workbench		This module explains how to test your applications with IBM Worklight Foundation.

Table 12. Moving to production

Module	Sample (if any)	Description
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.
Highly revised in 6.2.0: Operational Analytics		This module explains how you can use the operational analytics features of IBM Worklight Foundation 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.
Introducing Worklight Server and Application Center on IBM PureApplication System		This module explains how you can integrate 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.

Samples

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

Table 13. Worklight Starter applications

Sample	Description
Worklight Starter application	This file contains the sample code of the Worklight Starter application.
Worklight Starter application with jQuery Mobile	This file contains the sample code of the Worklight Starter application with jQuery Mobile.
Worklight Starter application with Sencha	This file contains the sample code of the Worklight Starter application with Sencha Touch.
Worklight Starter application with Dojo Mobile	This file contains the sample code of the Worklight Starter application with Dojo Mobile.
NEW in 6.2.0: Worklight Starter application for iOS	This file contains the sample code of the Worklight Starter application for iOS; to be used with the Worklight Starter application.
NEW in 6.2.0: Worklight Starter application for Android	This file contains the sample code of the Worklight Starter application for Android; to be used with the Worklight Starter application.

You can also study the following sample to learn how to use IBM Worklight Foundation to develop applications that are based on Dojo. This sample has an associated module that describes it.

Table 14. Running the Dojo-based sample

Module	Sample	Description
Running the Dojo-based sample	Worklight project	This module and its companion sample show how to develop an application that is based on Dojo through a basic sample application.

Additional resources

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

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

Terms and conditions

Use of the IBM Worklight Foundation 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 5. Known limitations

General limitations apply to IBM Worklight Foundation 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 Foundation 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 Foundation V6.2.0, 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.
- The console of the Worklight Server Configuration Tool has the same globalization limitation as the Windows console to display strings that are out of the default codepage.

IBM Worklight Analytics Platform has the following limitations in terms of globalization:

- In reports, the format for dates and times do not follow the International Components for Unicode (ICU) rules.
- In reports, the format for numbers 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 that are created with the WL.Analytics `log` method are not always logged successfully.
- Messages that include non-ASCII characters that are created with the WL.Logger `error` method 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 Characters that are valid for user IDs and passwords.
- 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.
- In Java 7.0 Service Refresh 4-FP2 and previous versions, you cannot paste Unicode characters that are not part of the Basic Multilingual Plane into the input field. To avoid this issue, create the path folder manually and choose that folder during the installation.
- Custom title and button names for the alert, confirm, and prompt methods should be kept short to avoid truncation at the edge of the screen.
- The applications that are developed with IBM Worklight Application Framework running in Portuguese (Portugal) will see runtime messages in Portuguese (Brazil).

- JSONStore does not handle normalization. All the Find functions for the JSONStore API does not take account of language sensitivity such as accent insensitive, case insensitive, and 1 to 2 mapping.
- The sorted results of JSONStore Find API are not language-specific and not compliant with Common Locale Data Repository (CLDR) rules.

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.

Note: The Rich Page Editor supports jQuery mobile 1.4.2. It does not support jQuery mobile 1.4.1

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 Worklight Analytics Platform clustering. Consider testing your load prior to integrating IBM Worklight Analytics Platform 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 Chapter 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343 or "Upgrading to Worklight Server V6.2.0 in a production environment" on page 355.

FIPS 140-2 feature limitations

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

- 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. FIPS is not yet supported on 64-bit architecture even though Worklight library does support 64-bit architecture. Therefore, FIPS must not be enabled on 64-bit target platform when XCode Build Setting (Architecture) is also set to 64 bit.
- This feature works with hybrid applications only (not native).
- 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 IBM Worklight V6.0 and before, under category 5, *Advanced client side development*.
  - The article *Enable FIPS 140-2 HTTPS encryption for IBM Worklight mobile apps* at Enable FIPS 140-2 HTTPS encryption for IBM Worklight mobile apps.

  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 occurs over a non-FIPS 140-2 protected channel. If you want to use only a FIPS 140-2 protected channel, you must disable the Direct Update feature. To disable the Direct Update feature, you can override the mobile security test in the `authenticationConfig.xml` file in your `server/conf` directory.

```
<mobileSecurityTest name="mobileTests">
  <testDirectUpdate mode="disabled" />
</mobileSecurityTest>
```

Use this mobile security test for Android and iOS environments as follows:

```
<android securityTest="mobileTests" version="1.0">
```

or

```
<iphone bundleId="com.TestHybridApp" version="1.0" securityTest="mobileTests">
```

Support for Android Emulator 2.3.x

IBM Worklight Foundation 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 "Creating hybrid applications with IBM Worklight Application Framework" on page 559, to your Worklight project or application. You use this feature for rapid development of hybrid Worklight applications. With IBM Worklight Foundation V6.2.0, 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 Foundation:

- 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.
- This feature cannot be used when you use the IBM HTTP Server between the client device and the Worklight Server. The IBM HTTP Server is unable to provide user identity to the Worklight Server after authenticating with the client certificate.

LTPA token limitations

An `SESN0008E` exception occurs when an LTPA token expires before the user session expires.

An LTPA token is associated with the current user session. If the session expires before an LTPA token expires, a new session is created automatically. However, when an LTPA token expires before a user session expires, the following exception occurs:

```
com.ibm.websphere.servlet.session.UnauthorizedSessionRequestException: SESN0008E: A user authenticate
```

To resolve this limitation, you must force the user session to expire when the LTPA token expires.

- On WebSphere Application Server Liberty, set the `httpSession` attribute `invalidateOnUnauthorizedSessionRequestException` to `true` in the `server.xml` file.
- On WebSphere Application Server, add the session management custom property `InvalidateOnUnauthorizedSessionRequestException` with the value `true` to fix the issue.

Note: On certain versions of WebSphere Application Server or WebSphere Application Server Liberty, the exception is still logged, but the session is correctly invalidated. For more information, see APAR PM85141.

Command Line Interface for IBM Worklight Developers fails when IPv6 is enabled

A `404 Not Found` error occurs when you start the server with the IBM Worklight command-line interface (CLI) and IPv6 addressing is set.

When you work with the IBM Worklight CLI, you might experience problems when you deploy the Worklight Server. These problems might occur if your environment has IPv6 addressing enabled and set or if the Worklight Server starts with IPv6 when it starts, by default, with IPv4.

When you deploy the Worklight Server with the IBM Worklight CLI and the IPv6 address is both enabled and set, you receive error messages. The following message is an example of these error messages:

```
[wladm] Error accessing http://10.0.0.1:10080/worklightadmin/management-apis/1.0/runtimes/wlproje
```

When you run the Worklight Server by using the IBM Worklight CLI and the IPv6 address is both enabled and set, the server starts with IPv6 instead of IPv4. The following messages are examples of the message that you might receive:

```
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKZ0001I: Application worklightadmin started in 12.274 seconds.
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKZ0001I: Application wlproject started in 28.558 seconds.
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKZ0001I: Application worklightconsole started in 13.893 seconds.
[AUDIT] CWWKF0011I: The server worklight is ready to run a smarter planet.
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKZ0001I: Application _analytics started in 4.236 seconds.
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
[AUDIT] CWWKZ0001I: Application _MobileBrowserSimulator started in 0.042 seconds.
[AUDIT] CWWKT0016I: Web application available (default_host): http://[a000:b000:c000:d000:a000:b00
```

This problem is caused by the Worklight Server starting with IPv6, and the IBM Worklight CLI attempts to deploy against an IPv4 server.

You can fix the problem by completing the following steps:

1. Open the directory for the Worklight Server, which is /[USER_HOME]/ .worklight/.
2. Move to the directory for the default server, which is /6.2.0/server/wlp/usr/ servers/worklight/.
3. Edit the `jvm.options` file that is located inside the opened directory.

- Add the following option to the end of this file:
  `-Djava.net.preferIPv4Stack=true`
- Save and close the file.

4. Start or run the Worklight Server to verify that the problem is fixed.

Chapter 6. Troubleshooting

You can find advice on how to troubleshoot problems, and more information about known limitations and technotes (Troubleshooting).

The following links point to troubleshooting topics in other parts of this user documentation. To navigate from there back to this topic, click **Back** in your Web browser.

- "Troubleshooting IBM Mobile Test Workbench for Worklight" on page 69
- "Troubleshooting Worklight Server" on page 337
- "Troubleshooting an installation blocked by DB2 connection errors" on page 338
- "Troubleshooting failure to create the DB2 database" on page 338
- "Troubleshooting IBM HTTP Server startup" on page 323
- "Troubleshooting to find the cause of installation failure" on page 338
- "Troubleshooting a Cast Iron adapter – connectivity issues" on page 738
- "JSONStore troubleshooting" on page 798
- "Troubleshooting adding and removing application components" on page 702
- "Troubleshooting the User Certificate Authentication feature" on page 1463
- "Simple data sharing troubleshooting" on page 931.
- "Troubleshooting a corrupt login page (Apache Tomcat)" on page 1315
- "Troubleshooting the views created with IBM Worklight Application Framework" on page 585
- "Troubleshooting failure to authenticate to Application Center and applications that use the basic registry element" on page 340
- "Troubleshooting unintended direct update requests on Windows Phone 8" on page 613

Important: For more information about known limitations or issues in the product, see Chapter 5, "Known limitations," on page 47 and the product Release notes.

Important: If you have to contact IBM Support for help, see the information in Collect troubleshooting data for IBM Worklight and IBM Mobile Foundation 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 7. Installing and configuring

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

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

IBM Worklight Foundation 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 58.

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 and configuring IBM Mobile Test Workbench for Worklight" on page 68.

Installing Worklight Server with IBM Installation Manager

To ensure the correct installation of Worklight Server, see "Installation prerequisites" on page 71.

You must install IBM Installation Manager 1.6.3 or later separately before installing IBM Worklight Foundation. For more information, see "Running IBM Installation Manager" on page 84.

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 70.

Upgrading IBM Worklight

The preceding sections provide an overview of IBM Worklight Foundation "first time" installations. For information about upgrading existing installations of Worklight Studio and Worklight Server to a newer version, see Chapter 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343.

Installing Worklight Studio

You install Worklight Studio from your existing Eclipse IDE workbench.

Before you begin
- Ensure that your computer meets the system requirements for the software that you install, as detailed in "System requirements for using IBM Worklight Foundation" on page 13. In particular, note the required versions of Eclipse (errors might occur if you use other versions or editions).
- If you anticipate using IBM Dojo Toolkit V1.10.1 that is available as part of Worklight Studio, ensure that Eclipse V4.4.1 or higher is installed.
- For Android development, ensure that Android SDK and Oracle JDK are installed.
- 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

Worklight Studio is installed with a P2 Eclipse update.

As of V6.2.0, IBM Worklight Developer Edition, IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition provide the same instance of Worklight Studio:
- Worklight Studio that is installed from the IBM Worklight Developer Edition now includes the authenticity feature, which was previously only available in the IBM Worklight Consumer Edition and the IBM Worklight Enterprise Edition. Developers can now create production-ready applications with any edition of Worklight Studio.
- The internal Worklight Development Server that is embedded in Worklight Studio does not include the authenticity feature. Deploying and testing the authenticity feature requires IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition. You must install Worklight Server, configure it in Worklight Studio as an external server, and deploy the application to Worklight Server.

To learn more about the specifics of IBM Worklight Developer Edition, IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition, see "Worklight editions" on page 12.

To install Worklight Studio as part of IBM Worklight Developer Edition, go to the IBM Worklight Foundation development website at https://www.ibm.com/developerworks/mobile/worklight.html.

To install Worklight Studio as part of IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition, complete the following procedure.

Procedure

1. Start your Eclipse IDE workbench. Verify that your version of Eclipse is one of the versions that is listed in "System requirements for using IBM Worklight Foundation" on page 13.
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 to the location of the archive file on your machine, and click **Open**.
6. Click **OK** to close the Add Repository window.
7. On the Available Software page, select **IBM Worklight Studio Development Tools**, and then 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**. The installation process starts.
10. 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.

You can now optionally install IBM Mobile Test Workbench for Worklight. For more information, see "Installing and configuring IBM Mobile Test Workbench for Worklight" on page 68.

Running additional tasks for Rational Team Concert V4.0

Before you run Worklight Studio, you might need to clean the Eclipse environment.

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\`.
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 after the existing lines, 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.

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 Adobe website at http://www.adobe.com/.
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 4, "Tutorials and samples," on page 35.

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 4, "Tutorials and samples," on page 35.

       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), click **Browse**, and specify the directory where your Android Studio is installed.
    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 from 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 in 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`.

   `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 4, "Tutorials and samples," on page 35.

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 4, "Tutorials and samples," on page 35.

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. Select the option to download the Windows SDK.
3. Download Microsoft Visual Studio Express 2012 from the Visual Studio website at http://www.visualstudio.com/ 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. 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.

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.

Uninstalling Worklight Studio

You uninstall Worklight Studio from your existing Eclipse IDE workbench.

About this task

Follow this procedure to uninstall Worklight Studio and the optional IBM mobile libraries, if those libraries were also installed on your computer along with Worklight Studio.

Procedure

1. In Eclipse, go to **Help** > **About Eclipse SDK** > **Installation Details**.
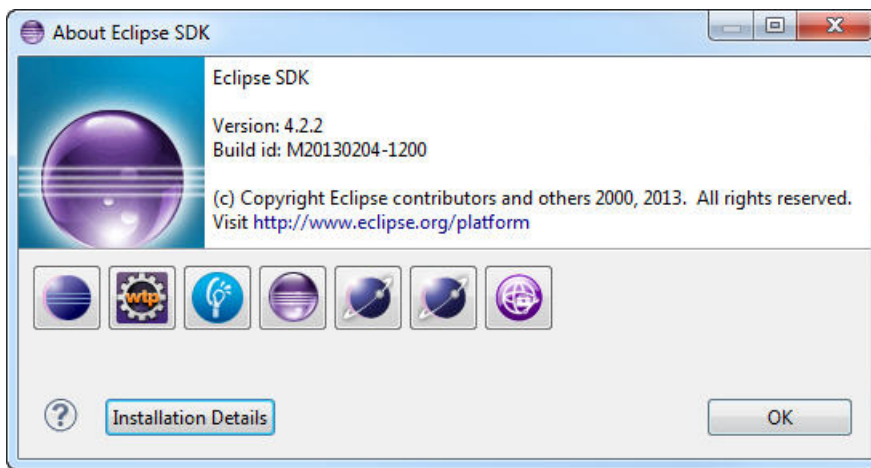
Figure 2. **About Eclipse SDK**

2. In **Eclipse SDK Installation Details**, select **IBM Worklight Studio**, and **IBM Dojo Mobile Tools** and **IBM jQuery Mobile Tools** if those components are also listed in the list of **Installed Software**.
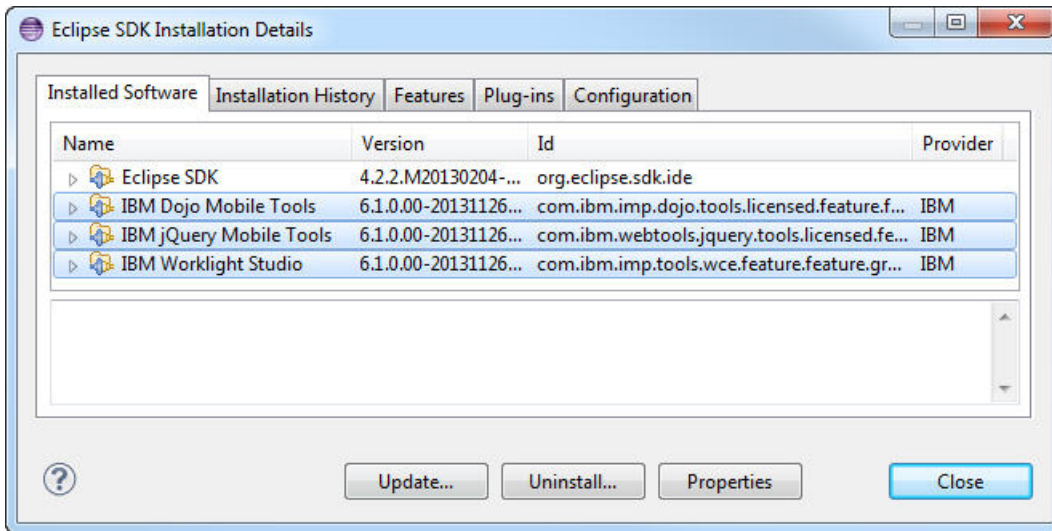
Figure 3. List of installed software in **Eclipse SDK Installation Details**

3. Click **Uninstall**.

4. In the **Uninstall** wizard, review the components to uninstall, and click **Finish**.

5. Wait until the uninstallation process finishes, and click **Yes** when you are prompted to restart Eclipse to complete the uninstallation.

Installing command-line tools for developers

Follow these instructions to install Command Line Interface for IBM Worklight Developers.

Procedure

1. Download `worklight_cli_installer_6.2.0.zip` from DeveloperWorks. For information about this file, see DeveloperWorks for IBM Worklight.

2. Open your command-line terminal and change the directory to the folder where your Worklight CLI package is located.

3. Then, run: `$ unzip worklight_cli_installer_6.2.0.zip`. The CLI is packaged as a single compressed file, which contains installation executable files for each platform, as listed:

   - Readme file
   - `install_linux.bin`
   - `install_mac.app`
   - `install_windows.exe`
   - `resources/`

4. Select and run the installer that is appropriate to your platform. For example, when you run `$ install_linux.bin` a GUI appears which guides you through the installation of Command Line Interface for IBM Worklight Developers. Follow the instructions to complete your installation.

5. On completion of your installation, log out and then log back in. This action ensures that the **worklight** and **wl** commands are on your system path.

Installing in silent mode

Before you can use the unattended installation method of installing the CLI, a response file is required.

Before you begin

To complete an unattended installation, you must use a previously created response file.

About this task

If you have more than one system to install and you already know which installation options you want, you can use the unattended installation process.

First, you must complete one attended installation, by using the GUI or Console installation. After you complete that first attended installation, the `installer.properties` response file is created in the same folder as the installer. This file contains the options for the attended installation. You can reuse this response file for subsequent installations that use the same options.

Note: The `installer.properties` file is overwritten each time you start a new installation. To prevent the file from being overwritten, you can rename the file.

Procedure

To use the response file and proceed with the silent installation:
1. Change the license accepted value to **TRUE**.
2. Remove the leading # from one of the **USER_INSTALL_DIR** properties.
3. Change the value to the location where you want to install.
   ```
   # Indicate whether the license agreement has been accepted
   #----------------------------------------------------
   LICENSE_ACCEPTED=FALSE

   # Choose Install Folder
   # Uncomment one of the properties below, and (optionally) change the value
   # if you prefer to install to a non-default location.
   #---------------------
   # Linux default
   #USER_INSTALL_DIR=/opt/ibm/Worklight-CLI

   # Windows default
   #USER_INSTALL_DIR=C:\\Program Files\\IBM\\Worklight-CLI

   #---------------------
   ```
4. Install the **CLI** with the silent installation by writing one of the following command lines.
   - Windows: `install_windows.exe` **-i** `silent` **-f** `/path/to/ installer.properties`
   - Linux: `./install_linux.bin` **-i** `silent` **-f** `/path/to/installer.properties`

Windows 8.1 installation for Command Line Interface for IBM Worklight Developers

If you are using Windows 8.1, you need to ensure that the installer is run in **compatibility mode**.

Procedure

Follow these steps to run the installer in compatibility mode.
1. Right click `install_windows.exe` and click **Properties**:
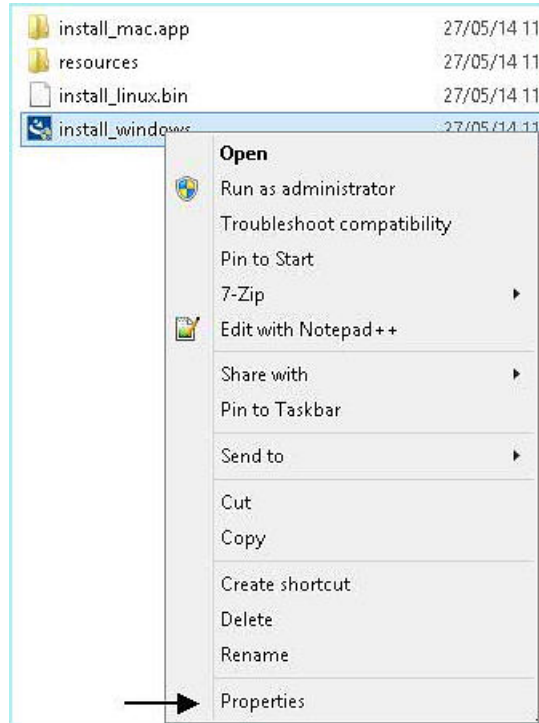
2. You can see the **Properties** window, click the **Compatibility** tab.
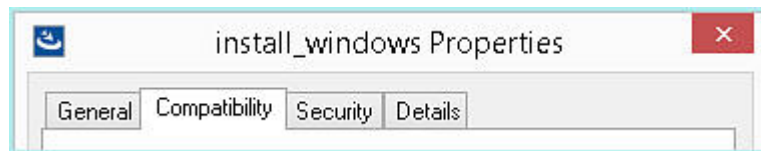
3. Check the option **Run this program in compatibility mode for**:
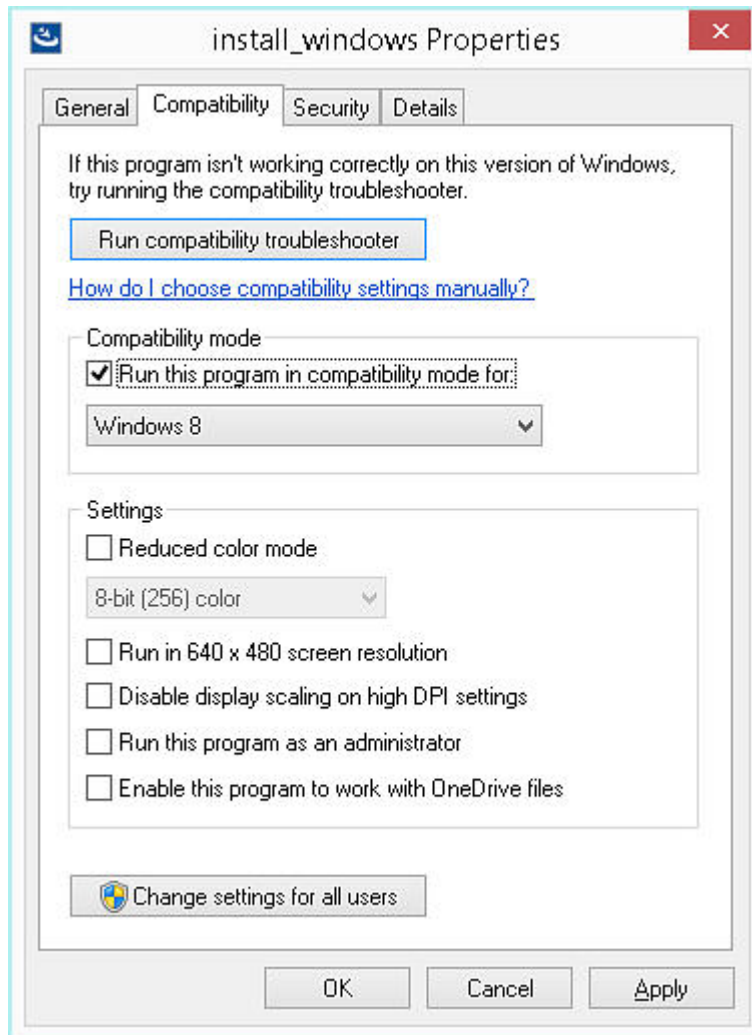
4. Click **Apply** and then **OK**.
5. Then, double-click `install_windows.exe`.
6. The installer opens and works as expected.

Console Installation for Command Line Interface for IBM Worklight Developers

Follow these instructions to install Command Line Interface for IBM Worklight Developers.

Procedure

1. Download `worklight_cli_installer_6.2.0.zip` from DeveloperWorks. For information about this file, see DeveloperWorks for IBM Worklight.
2. Open your command-line terminal and change the directory to the folder where your extracted Worklight CLI package is located.
3. Select the installer that is appropriate to your platform and run the installer by using the next command line:
   - Windows: `install_windows.exe` **-i** `console`
   - Linux: `/install_linux.bin` **-i** `console`

4. A new command-line terminal window appears which guides you through the installation of Command Line Interface for IBM Worklight Developers. Follow the instructions to complete your installation.

5. On completion of your installation, log out and then log back in. This action ensures that the **worklight** and **wl** commands are on your system path.

Uninstalling command-line tools for developers

Follow these instructions to uninstall the Command Line Interface for IBM Worklight Developers.

Before you begin

Open your command-line terminal to the path where you installed the Command Line Interface for IBM Worklight Developers, and change the directory to the `Uninstaller` folder.

Procedure

1. **GUI Uninstallation:** Select and run the uninstall. A GUI appears which guides you through the uninstallation of the Command Line Interface for IBM Worklight Developers. Follow the instructions to complete your uninstallation.

   Note: Mac OS X users can only uninstall the product with the GUI.

2. **Console Uninstallation:** Run the uninstall by using the appropriate command line:
   - Windows: `Uninstall.exe -i console`
   - Linux: `./Uninstall -i console`

   A new command-line terminal window appears which guides you through the uninstallation of the Command Line Interface for IBM Worklight Developers. Follow the instructions to complete your uninstallation.

3. **Silent Uninstallation:** Run the uninstall by using the appropriate command line:
   - Windows: `Uninstall.exe -i silent`
   - Linux: `./Uninstall -i silent`

Installing and configuring 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 Foundation" on page 13.
- 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 Worklight Studio as part 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 Worklight Studio as part of IBM Worklight Consumer Edition or of 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 Android and the mobile test client for the iOS Simulator. For more information about the mobile test client installation, see Installing the mobile test client.

You can then configure the mobile test client. For details, see Configuring the mobile test client.

To understand the various available features, see Chapter 10, "Testing with IBM Worklight Foundation," on page 987.

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

Troubleshooting IBM Mobile Test Workbench for Worklight

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

About this task

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

Procedure

1. In Eclipse, go to **Window** > **Preferences** > **Java** > **Installed JREs**.
2. Select **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 Foundation.

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 a 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 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343.

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 "Tutorial for a basic installation of Worklight Server" on page 77 are based on a simple installation scenario. For a complete description of the process of deploying a Worklight project, see Chapter 12, "Deploying Worklight projects," on page 1101.

Planning the installation of Worklight Server

You must plan your installation and choose one installation scenario. You must also plan the creation of your databases and the topology of the application server.

To install the Worklight Server, you can choose one of the following scenarios:

- With the Server Configuration Tool.

  The Server Configuration Tool is a graphical tool and is available for Windows, Linux on x86, and Mac OS. With this tool, you get easily started, but expect some limitations when you maintain an application in production, in particular for some upgrade scenarios. This tool can export Ant files.

  Restriction:

  – The Server Configuration Tool does not support server farms. Therefore, you cannot use it to define, install, upgrade, or uninstall server farms. For server farms, use the provided Ant script or follow manual steps in your application server. For more information, see "Installing a server farm" on page 164.
  – The Server Configuration Tool for Mac OS is available for development and test purposes only.

- Ant tasks: Ant command-line files automate the process of creating or upgrading a database, either automatically or as a complement of a database preparation by a database administrator. The Ant tasks also automate the process of installing or upgrading the Administration Services and the Worklight Console in an application server. Ant tasks provide a high level of control for individual operations on the database or on the application server.
- Manual installation.

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 Foundation" on page 13.

Important: If a version of Worklight Server is already installed, review Chapter 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343 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 Foundation package from IBM Passport Advantage.

Ensure that you have the latest fix packs for the IBM Worklight Foundation 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 Worklight Server.

The 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 you start the Worklight Server installer. However, you need them before you deploy IBM Worklight Foundation projects.

The IBM Worklight Foundation packages include the following installers:
- IBM DB2 Workgroup Server Edition
- IBM DB2 Enterprise Server Edition (on Linux for System z® only)
- IBM WebSphere Application Server Liberty Core

File system prerequisites
To install IBM Worklight Foundation to an application server, the Worklight installation tools must be run by a user that has specific file system privileges.

The installation tools include:
- IBM Installation Manager
- The Server Configuration Tool
- The Ant tasks to deploy the Worklight Server

For WebSphere Application Server Liberty profile, you must have the right to perform the following actions:
- Read the files in the Liberty installation directory.
- Create files in the configuration directory of the Liberty server, which is typically usr/servers/*<servername>*, to create backup copies and modify server.xml and jvm.options.
- Create files and directories in the Liberty shared resource directory, which is typically usr/shared.
- Create files in the Liberty server apps directory, which is typically usr/servers/<servername>/apps.

For WebSphere Application Server full profile and WebSphere Application Server Network Deployment, you must have the right to perform the following actions:
- Read the files in the WebSphere Application Server installation directory.
- Read the configuration file of the selected WebSphere Application Server full profile or of the Deployment Manager profile.
- Run the **wsadmin** command.
- Create files in the profiles configuration directory. The installation tools put resources such as shared libraries or JDBC drivers in that directory.

For Apache Tomcat, you must have the right to perform the following actions:
- Read the configuration directory.
- Create backup files and modify files in the configuration directory, such as server.xml, and tomcat-users.xml.
- Create backup files and modify files in the bin directory, such as setenv.bat.
- Create files in the lib directory.
- Create files in the webapps directory.

For all these application servers, the user who runs the application server must be able to read the files that were created by the user who ran the Worklight installation tools.

Introduction to the Worklight Server components

The Worklight Server is composed of one or more runtime environments, an administration console and administration services, an enterprise application store, and an operational analytics feature.

Worklight Server components run as web applications on an application server. To set up Worklight Server, you must first choose which of the following supported application servers you want to use:

- WebSphere Application Server
- WebSphere Application Server Liberty profile
- Apache Tomcat

For implementations that require several servers, you might want to set up one of the following topologies:

- A farm of individual application servers. Specific installation steps must be taken. See "Installing a server farm" on page 164.
- A server cluster that is defined by using the WebSphere Application Server Network Deployment solution. (Use the hardware-sizing tool on developerWorks to decide on the number of servers and the size of the database.)

The following sections describe each of the following Worklight Server components:

- "Worklight runtime environments."
- "Worklight Console and Administration Services"
- "Application Center" on page 74
- "Worklight Operational Analytics" on page 74

Worklight runtime environments

The Worklight runtime environment is a mobile-optimized server-side component that runs the server side of your mobile applications (back-end integration, version management, security, unified push notification).

Each runtime environment is packaged as a web application (WAR file), which is created by using Worklight Studio. Each runtime environment can host one or more Worklight applications.

Each runtime environment requires a database to host information such as the list of devices that connect to it. Different runtime environments (different project WAR files) cannot share database tables. If you have multiple runtime environments, the tables must be in different schemas or different databases, depending on your database management system. Similar runtime environments (project WAR files) that run in the same cluster of farm must use the same database. You can also optionally set up a report database to host data for simple usage reports of a particular runtime environment.

Worklight Console and Administration Services

The Worklight Administration component consists of two applications that are used for the administration of the runtime environments:

- The Worklight Console application.
- The Worklight Administration Services application.

The Worklight Console is the web-based interface that an IT administrator uses to run administration tasks on the mobile application such as application deployment, management, version enforcement, and management of push notifications.

The Worklight Console is supported by the Worklight Administration Services application. This application acts as host for all the REST services and administration tasks. Both the Worklight Console and the Administration Services can be secured through standard Java Platform, Enterprise Edition security. Several administration user roles are available to cover different administration scenarios.

One Worklight Console can be used for the administration of several runtime environments. The administration tasks are run through standard JMX (Java Management Extension) calls.

Note: Since V6.2.0, the Worklight runtime environment registers an MBean called `com.worklight.core.jmx.ProjectManagementMXBean` and exposes the JMX API with this MBean. This MBean is exposed with all other MBeans that are registered in the application server. It is used by the runtime environment for such things as the management of deployed applications and adapters, devices, and push services. This MBean is intended to be used only by the Worklight Administration Services, so methods in this MBean are private and are not meant to be accessed directly.

A database is required for the Worklight Console and Administration Services.

Application Center

The Application Center is a private application store that developers can use to get feedback about their mobile applications from testers and stakeholders. The Application Center is used to distribute private or public applications to employees and partners. The Application Center is an optional component that can be installed separately from the other Worklight Server components.

The Application Center consists of two web applications:
- The Application Center console application that provides the user interface.
- The Application Center services application that provides the REST services.

A database is required for the Application center. For more information, see "Application Center" on page 1295.

Worklight Operational Analytics

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

The feature is packaged as a web application. To enable the collection of analytics information, you must configure the runtime environment with the URL of the analytics platform. For more information, see "Analytics" on page 1395.

Planning the creation of the databases

You must plan the creation of the three databases that are needed for the Administration Services and the Worklight runtime environments.

The installation of the Worklight Server requires the following three databases:
- For the Administration Services, an administration database.
- For each Worklight runtime environment:

- a database for Worklight
- a database for Worklight Reports

Optionally, the Application Center can be installed; the Application Center also requires a database.

An installation of Worklight Server includes at least one Worklight runtime environment, which is the web application that is in contact with the mobile devices, but might contain more than one Worklight runtime environment.

The databases can be instantiated automatically by the Server Configuration Tool or by the Ant tasks. In these two installation scenarios, it is also possible that a database administrator creates the database beforehand. For more information about the creation of these databases, see "Optional creation of the administration database" on page 104. For more information about the Worklight runtime environments, see "Optional creation of databases before you use the Ant tasks" on page 1110.

For DB2, the administration, the Worklight, and Worklight Reports databases can be in the same database, but they must be in different schemas.

For Oracle, these databases must be created for a different user.

For each database, it is possible to restrict the privileges of the database user that uses the data source at run time.

Restricting database user permissions for Worklight Server runtime operations:

When the databases are operational, you can decide to create a database user with restricted privileges. You use this database user to perform database underlying operations from the Worklight administration and runtime components. The user credentials appear in the application server configuration.

Worklight Server data is stored in three databases, which are described in "Introduction to the Worklight Server components" on page 73. The database administrator might require you to provide specific permissions that you need when you access those databases at run time. The connection to the Worklight Server databases at run time, which is established in the data source credentials, and any subsequent requests to the databases, are handled through a single database user or one distinct user per database. Using different users that can access only one kind of database, and especially to separate the databases of the Worklight runtime environment from the database of the Worklight administration component, improves security. These database users have no relation to the standard Worklight Server groups. The following table shows the minimal permissions that the database administrator must define on the Worklight Server databases for these users:

Table 15. Minimal permissions defined by the database administrator

Database permission	Use Worklight Server Operation
ALTER TABLE	Not required
CREATE INDEX	Not required
CREATE ROLE	Not required
CREATE SEQUENCE	Not required
CREATE TABLE	Not required

Database permission	Use Worklight Server Operation
CREATE VIEW	Not required
DROP INDEX	Not required
DROP SEQUENCE	Not required
DROP TABLE	Not required
DROP VIEW	Not required
SELECT TABLE	Required
INSERT TABLE	Required
UPDATE TABLE	Required
DELETE TABLE	Required
SELECT SEQUENCE	Required

These minimal permissions also apply to the database user of the (optional) Application Center database.

Using complex Oracle connection descriptors:

For some topologies of Oracle, for example Oracle Real Application Clusters (RAC), you might have to use complex Oracle Net connection descriptors. In that case, review the following steps.

Procedure
1. You must create the databases manually for the Application Center, the Worklight Server administration, and the Worklight project WAR file. This step is mandatory and cannot be performed with the Ant tasks or Server Configuration Tool. See the following links for instructions on how to create these databases.
   - For installing the Application Center, see "Creating the Oracle database for Application Center" on page 240.
   - For installing the Worklight Server administration, see "Creating the Oracle database for Worklight Server administration" on page 106.
   - For deploying the Worklight project WAR file, see "Creating the Oracle databases" on page 1112.
2. In IBM Installation Manager, or in the Worklight Server Configuration Tool, you must use a generic Oracle JDBC URL instead of the host name and port.

Figure 4. Oracle Database Settings window

3. For Ant tasks, you must use the alternative attributes for the `<oracle>` element. For more information, see "Ant **configuredatabase** task reference" on page 1115, table 19.

   Note: The example files in "Sample configuration files" on page 1156 do not use the alternative attributes for the `<oracle>` element. If you use an example file, you must modify the `<oracle>` elements in the file so that they use the alternative attributes.

4. The URL must be a URL for the Oracle thin driver. It must not include the user name and password, for example: `jdbc:oracle:thin:@(DESCRIPTION= [Oracle Net connection descriptor])`.

Planning the topology of the application server

You must install Worklight Server in an application server, and decide which topology to use.

For more information about the choice of topology, see "Typical topologies of a Worklight instance" on page 303.

Tutorial for a basic installation of Worklight Server

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 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 tutorial 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 a 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 simple Worklight project and its console.

Procedure

1. Install WebSphere Application Server Liberty Core. The installer for WebSphere Application Server Liberty Core is provided as part of the package for IBM Worklight Foundation.

   a. Load the repository for 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 file names 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`.

      For more information about loading repositories with IBM Installation Manager, see step 4a of this procedure. See also the IBM Installation Manager user documentation at https://www.ibm.com/support/knowledgecenter/SSDV2W_1.7.0/com.ibm.cic.agent.ui.doc/helpindex_imic.html.

   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 a 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 tutorial.

3. Install the database management system. You use this DBMS to install the Application Center and to deploy a 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 Foundation. 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 following 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 user name and password for the DB2 administrator role.

b. If you use MySQL:

1) Install MySQL on your computer.

2) Take note of the user name 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) Start the `ssh` server (on Windows, with administrator rights).

4) In the following steps, you must have that Secure Shell server running.

4. Install Worklight Server.

a. Add the Worklight Server repository in IBM Installation Manager:

1) Download the **Installation Manager Repository for IBM Worklight Server** from Passport Advantage.

2) Extract the file on your disk.

3) Start 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 extracted.

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 tutorial.

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 (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 100 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 264.
      - See "Distribution structure of Worklight Server" on page 100 for a list of 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 243. 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. Install the Worklight Server administration components: Administration Services and Worklight Console.

   a. Start the 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 Worklight Server Configuration**.

   c. Name the configuration `Hello Worklight Server`.

   d. Do not change the default entries in the **Configuration Description** panel.

   e. Do not change the default entries in the **Console Settings** panel.

   f. 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.

g. 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.

- Take note of the default password and login: demo (for both).

h. 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 that is named databases appears in the tree view, under the **Configuration**.

- After the deployment to the application server is complete, a log file that is named install appears in the tree view, under the **Configuration**.

7. Create a simple Worklight project. You create a Worklight runtime environment.

a. Complete the following steps:

1) Install Worklight Studio on your computer. See "Installing Worklight Studio" on page 58.

2) Start Worklight Studio.

3) Create a 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 step 8. 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.

8. Deploy a Worklight runtime environment with the Server Configuration Tool.

a. In the Server Configuration Tool, select **File/Add Worklight runtime environment**

b. In the dialog box, select the **Hello Worklight Server** configuration created in step 6.

c. In **Enter the name of the new runtime**, enter First Runtime.

d. In the **General** panel:

- Load the WAR file that you created in the previous step.

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. When all the data is entered, click **Deploy**.

- The log of the deployment operations appears in the console.

- The **Runtime** appears in the tree view.
- After the database operation is completed, a log file that is named `databases` appears in the tree view, under the **Configuration**.
- After the deployment to the application server is complete, a log file that is named `install` appears in the tree view, under the **Configuration**.

9. 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 see the Worklight Console. You can log in with the default user login and password that you created in step 6 (by default `demo/demo`).

What to do next

For more information about the Worklight Server Configuration Tool, see "Deploying, updating, or undeploying Worklight Server by using the Server Configuration Tool" on page 1106.

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 1195.
- Review Chapter 13, "Administering Worklight applications," on page 1247.
- Review "Deploying the project WAR file" on page 1105.
- Review "Configuration of Worklight applications on the server" on page 1171 and "Configuring a Worklight project in production by using JNDI environment entries" on page 1182
- Review the options to deploy an IBM Worklight Foundation 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 1163.

Running IBM Installation Manager

IBM Installation Manager installs the IBM Worklight Foundation 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 that 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 Foundation in several different modes, including single-user and multi-user installation modes.

You can also use silent installations to deploy IBM Worklight Foundation to multiple systems, or systems without a GUI interface.

For more information about Installation Manager, see the IBM Installation Manager user documentation.

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 file names 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`.

Installation of the Application Center with IBM Installation Manager

Run IBM Installation Manager. If you plan to install the Application Center with IBM Installation Manager, verify that the user who runs IBM Installation Manager has the privileges that are described in "File system prerequisites" on page 72, then see "Installing and configuring the Application Center" on page 238 and install the Application Center before you proceed to the installation of Worklight Console. For more information, see "Installing and configuring Worklight Server administration" on page 104.

If you do not plan to install the Application Center with IBM Installation Manager, or if you plan to install the Application Center manually, answer **No** to the question **Install IBM Application Center**.

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 can 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 sets 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 afterward 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.

Note: Worklight Server does not support the group mode of IBM Installation Manager.

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 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343. It describes important steps to upgrade

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**.

Command-line installation with XML response files (silent installation)

With IBM Installation Manager, you can complete a command-line installation of Worklight Server with XML response files, on multiple computers, or on computers where a GUI interface is not available. In the following documentation, this installation is referred to as silent installation.

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 user documentation, see Working in silent mode.

There are two ways to create a suitable response file:

- Working with sample response files provided in the Worklight user documentation.
- 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 "Command-line (silent installation) parameters" on page 91.

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 use them.

1. Pick the appropriate sample response file from the compressed file. The `Silent_Install_Sample_Files.zip` file contains one subdirectory per release.

   Important: 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 16. Sample installation response files in the `Silent_Install_Sample_Files.zip` file to install Application Center

Application server where you install the Application Center	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`

Table 16. Sample installation response files in the `Silent_Install_Sample_Files.zip` file to install Application Center (continued)

Application server where you install the Application Center	Derby	IBM DB2	MySQL	Oracle

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`

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. You can use IBM DB2 or another database that is 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 that is named **IBM Worklight**, whereas Worklight Server V5.x used the package group **Worklight**).

Table 17. 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 368 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. Type the following command:
   ```
   <InstallationManagerPath>/eclipse/tools/imcl input <responseFile>  -log /tmp/installwl.log -ac
   ```

   Where:
   - `<InstallationManagerPath>` is the installation directory of IBM Installation Manager.
   - `<responseFile>` is the name of the file that is selected and updated in step 1.

   For more information, see 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.

Command-line (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 18. Parameters available for silent installation

Key	When necessary	Description	Allowed values
user.appserver.selection2	Always	Type of application server. `was` means preinstalled WebSphere Application Server 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	WebSphere Application Server installation directory.	An absolute directory name.
user.appserver.was.profile	${user.appserver.selection2} == was	Profile into which to install the applications. For WebSphere Application Server Network Deployment, specify the Deployment Manager profile. `Liberty` means the Liberty profile (subdirectory `wlp`).	The name of one of the WebSphere Application Server profiles.
user.appserver.was.cell	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	WebSphere Application Server cell into which to install the applications.	The name of the WebSphere Application Server cell.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
user.appserver.was.node	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	WebSphere Application Server node into which to install the applications. This corresponds to the current machine.	The name of the WebSphere Application Server 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 WebSphere Application Server Network Deployment cell. `nd-cluster` means a WebSphere Application Server Network Deployment cluster. `nd-node` means a WebSphere Application Server Network Deployment node (excluding clusters). `nd-server` means a managed WebSphere Application Server Network Deployment server.	`server`, `nd-cell`, `nd-cluster`, `nd-node`, `nd-server`

Table 18. 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 WebSphere Application Server server into which to install the applications.	The name of a WebSphere Application Server 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 WebSphere Application Server Network Deployment cluster into which to install the applications.	The name of a WebSphere Application Server Network Deployment cluster in the WebSphere Application Server 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 WebSphere Application Server Network Deployment node into which to install the applications.	The name of a WebSphere Application Server Network Deployment node in the WebSphere Application Server cell.
`user.appserver.was.nd.server`	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty && ${user.appserver.was.scope} == nd-server	Name of WebSphere Application Server Network Deployment server into which to install the applications.	The name of a WebSphere Application Server Network Deployment server in the given WebSphere Application Server Network Deployment node.
`user.appserver.was.admin.name`	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Name of WebSphere Application Server administrator.	
`user.appserver.was.admin.password`	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Password of WebSphere Application Server administrator, optionally encrypted in a specific way.	

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
user.appserver.was.appcenteradmin.password	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Password of appcenteradmin user to add to the WebSphere Application Server 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.
user.appserver.was85liberty.serverInstance	${user.appserver.selection2} == was && ${user.appserver.was.profile} == Liberty	Name of WebSphere Application Server 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.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
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
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.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`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.	
`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.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
`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 `${user.database.oracle.appcenter.jdbc.url}` is specified	The host name or IP address of the Oracle database server.	
`user.database.oracle.port`	`${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.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
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.username} if it starts with an alphabetic character and does not contain lowercase characters, otherwise it must be ${user.database.oracle.appcenter.username} 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.
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.

Table 18. Parameters available for silent installation (continued)

Key	When necessary	Description	Allowed values
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.

Distribution structure of Worklight Server

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

Table 19. Files and subdirectories in the Worklight Server installation directory

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

Table 20. Files and subdirectories in the WorklightServer subdirectory

Item	Description
worklight-jee-library.jar	The Worklight Server library for production. For instructions on deploying a Worklight project and this library to an Application Server, see Chapter 12, "Deploying Worklight projects," on page 1101. The deployment is typically performed by using Ant tasks, but instructions for manual deployment are also provided.
worklight-ant-deployer.jar	A set of Ant tasks that help you deploy projects, applications, and adapters to your Worklight Server. For documentation about the Ant tasks that are provided in this library, see Chapter 12, "Deploying Worklight projects," on page 1101.
worklight-ant-builder.jar	A set of Ant tasks that help you build projects, applications, and adapters for use in Worklight Server. For more information about the Ant tasks that are provided in this library, see Ant tasks for building and deploying applications and adapters.
configuration-samples	Contains the sample Ant files for configuring a database for the Worklight Server and deploying a Worklight project to an Application Server. For instructions on how to use these Ant projects, see "Sample configuration files" on page 1156.

Table 20. Files and subdirectories in the `WorklightServer` *subdirectory (continued)*

Item	Description
`databases`	SQL scripts to be used for the manual creation of tables for Worklight Server and the Administration Services, instead of using Ant tasks for the automatic configuration of these tables. For information about these scripts, see "Creating and configuring the databases manually" on page 1121.
`FarmSchemas.xsd`	XML schema that describes the format of the file that defines the nodes of a server farm.
`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. For information about this tool, see "Storing properties in encrypted format" on page 1178.
`report-templates`	Report templates to configure BIRT reports for your Application Server. For information about these BIRT reports, see "Manually configuring BIRT Reports for your application server" on page 1438.
`wladm-schemas`	XML schemas that describe the format of input and output of the `<wladm>` Ant task.
`worklightadmin.war`	The WAR file for the Administration Services web application.
`worklightconsole.war`	The WAR file for the Worklight Console user interface web application.
`external-server-libraries`	The JAR files and JavaScript libraries that allow you to use SSO between IBM Worklight Foundation and other external servers. See usage instructions in "Using SSO between IBM Worklight Foundation and external services" on page 917.

Table 21. 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 you use 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.

Table 21. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

Item	Description
ApplicationCenter/installer/IBMAppCenter	Contains the Worklight Studio project for the mobile Client. You must compile this project to create the iOS version of the mobile client.
ApplicationCenter/console/	**appcenterconsole.war** The WAR file for the Application Center console user interface web application. **applicationcenter.war** The WAR file for the Application Center REST services web application. **applicationcenter.ear** The enterprise application archive (EAR) file to be deployed under IBM PureApplication System.
ApplicationCenter/databases	**create-appcenter-derby.sql** The SQL script to re-create the application center database on derby. **create-appcenter-db2.sql** The SQL script to re-create the application center database on DB2. **create-appcenter-mysql.sql** The SQL script to re-create the application center database on mySQL. **create-appcenter-oracle.sql** The SQL script to re-create the application center database on Oracle. In addition, this directory contains the SQL scripts to upgrade the database from earlier versions of IBM Worklight Foundation.

Table 21. 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 22. 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 23. 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. For more information, see Chapter 12, "Deploying Worklight projects," on page 1101.

Table 24. Files and subdirectories in the `Analytics` subdirectory

Item	Description
`analytics.war`	Worklight Analytics Platform installation WAR file. For installation instructions, see "Installing the IBM Worklight Analytics Platform" on page 232.

Installing and configuring Worklight Server administration

You must install the Administration Services, and optionally the Worklight Console, as part of the Worklight Server installation.

Optional creation of the administration database

If you want to activate the option to install the administration database when you run the Ant tasks or the Server Configuration Tool, you need to have certain database access rights that entitle you to create the tables that are required by the administration database.

If you have sufficient database administration credentials, and if you enter the administrator user name and password in the installer when prompted, the installation tools can create the databases for you. Otherwise, you need to ask your database administrator to create the required database for you. In this case, the database must be created before you start the installation tools.

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

Important: This step is optional if you install IBM Worklight Foundation with the Server Configuration Tool or the Ant tasks because the Server Configuration Tool and the Ant tasks can create the databases automatically.

Creating the DB2 database for Worklight Server administration:

During the installation of IBM Worklight Foundation, the installation tools can create the administration database for you.

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the DB2 database manually for the Worklight Server administration" on page 127 instead.

About this task

The installation tools can create the administration 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 administration 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 WLADMIN) 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 Worklight Server 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 administration database, replacing the user name `wluser` with your chosen user names:

     ```
     CREATE DATABASE WLADMIN COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WLADMIN
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WLADMIN
     QUIT
     ```

3. The installation tools can create the database tables and objects for Worklight Server administration in a specific schema. This allows you to use the same database for Worklight Server administration 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 administration database tables and objects.

Creating the MySQL database for Worklight Server administration:

During the Worklight installation, the installer can create the administration database for you.

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the MySQL database manually for the Worklight Server administration" on page 135 instead.

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 WLADMIN) 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 WLADMIN CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON WLADMIN.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON WLADMIN.* 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 Foundation runs.

Creating the Oracle database for Worklight Server administration:

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

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the Oracle database manually for the Worklight Server administration" on page 138 instead.

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 WLADMIN. If you want multiple Worklight Server 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 system privilege: **CREATE SESSION**

       – Add system privilege: **CREATE SEQUENCE**

       – Add system privilege: **CREATE TABLE**

       – Add system privilege: **UNLIMITED TABLESPACE**

   • Using the `Oracle SQLPlus` command-line interpreter.

   The commands in the following example create a user named WLADMIN for the database:

```
CONNECT system/<system_password>@ORCL
CREATE USER WLADMIN IDENTIFIED BY password;
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WLADMIN;
DISCONNECT;
```

Configuration of the application server

IBM Worklight Foundation has some requirements for the configuration of the application server that are detailed in the following topics.

Configuring WebSphere Application Server Liberty profile:

You must configure a secure JMX connection for WebSphere Application Server Liberty profile.

Procedure

Worklight Server requires the secure JMX connection to be configured.

• The Server Configuration Tool and the Ant tasks can configure a default secure JMX connection, which includes the generation of a self-signed SSL certificate with a validity period of 365 days. This configuration is not intended for production use.

• To configure the secure JMX connection for production use, follow the instructions from the WebSphere Application Server user documentation.

• The rest-connector is available for WebSphere Application Server, Liberty Core, and other editions of Liberty, but it is possible to package a Liberty Server with a subset of the available features. To verify that the rest-connector feature is available in your installation of Liberty, enter the following command:

```
<libertyInstallDir>/bin/productInfo featureInfo
```

Note: Verify that the output of this command contains `restConnector-1.0`.

What to do next

For more information about the optimization of Worklight Server, especially the tuning of the JVM memory allocation, see "Optimization and tuning of Worklight Server" on page 181.

Configuring Apache Tomcat:

You must configure a secure JMX connection for Apache Tomcat application server.

About this task

The Server Configuration Tool and the Ant tasks can configure a default secure JMX connection, which includes the definition of a JMX remote port, and the definition of authentication properties. They modify *<tomcatInstallDir>*/bin/setenv.bat and *<tomcatInstallDir>*/bin/setenv.sh to add these options to `CATALINA_OPTS`:

```
-Djava.rmi.server.hostname=localhost
-Dcom.sun.management.jmxremote.port=8686
-Dcom.sun.management.jmxremote.authenticate=false
-Dcom.sun.management.jmxremote.ssl=false
```

Note: 8686 is a default value. The value for this port can be changed if the port is not available on the computer.

The `setenv.bat` file is used if you start Apache Tomcat with:
* *<tomcatInstallDir>*/bin/startup.bat, or *<tomcatInstallDir>*/bin/catalina.bat
* *<tomcatInstallDir>*/bin/startup.sh, or *<tomcatInstallDir>*/bin/catalina.sh

This file might not be used if you start Apache Tomcat with another command. If you installed the Apache Tomcat Windows Service Installer, the service launcher does not use `setenv.bat`.

Important: This configuration is not secure by default. To secure the configuration, you must manually complete steps 2 and 3 of the following procedure.

Procedure

Manually configuring Apache Tomcat:
1. For a simple configuration, add the following options to `CATALINA_OPTS`:
   ```
   -Djava.rmi.server.hostname=localhost
   -Dcom.sun.management.jmxremote.port=8686
   -Dcom.sun.management.jmxremote.authenticate=false
   -Dcom.sun.management.jmxremote.ssl=false
   ```
2. To activate authentication, see the Apache Tomcat user documentation SSL Support - BIO and NIO and SSL Configuration HOW-TO.
3. For a JMX configuration with SSL enabled, add the following options:
   ```
   -Dcom.sun.management.jmxremote=true
   -Dcom.sun.management.jmxremote.port=8686
   -Dcom.sun.management.jmxremote.ssl=true
   -Dcom.sun.management.jmxremote.authenticate=false
   -Djava.rmi.server.hostname=localhost
   -Djavax.net.ssl.trustStore=<key store location>
   -Djavax.net.ssl.trustStorePassword=<key store password>
   -Djavax.net.ssl.trustStoreType=<key store type>
   ```

```
-Djavax.net.ssl.keyStore=<key store location>
-Djavax.net.ssl.keyStorePassword=<key store password>
-Djavax.net.ssl.keyStoreType=<key store type>
```

Note: The port *8686* can be changed.

4. If the Tomcat instance is running behind a firewall, the JMX Remote Lifecycle Listener must be configured. See the Apache Tomcat documentation for JMX Remote Lifecycle Listener.

The following environment properties must also be added to the Context section of the Administration Services application in the server.xml file, such as in the following example:

```
<Context docBase="worklightadmin" path="/worklightadmin ">
    <Environment name="ibm.worklight.admin.rmi.registryPort" value="registryPort" type="java.l
    <Environment name="ibm.worklight.admin.rmi.serverPort" value="serverPort" type="java.lang.
</Context>
```

In the previous example:

- *registryPort* must have the same value as the rmiRegistryPortPlatform attribute of the JMX Remote Lifecycle Listener.
- *serverPort* must have the same value as the rmiServerPortPlatform attribute of the JMX Remote Lifecycle Listener.

Note: If you installed Apache Tomcat with the Apache Tomcat Windows Service Installer instead of adding the options to CATALINA_OPTS, run *<TomcatInstallDir>*/bin/Tomcat7w.exe, and add the options in the **Java** tab of the Properties window.
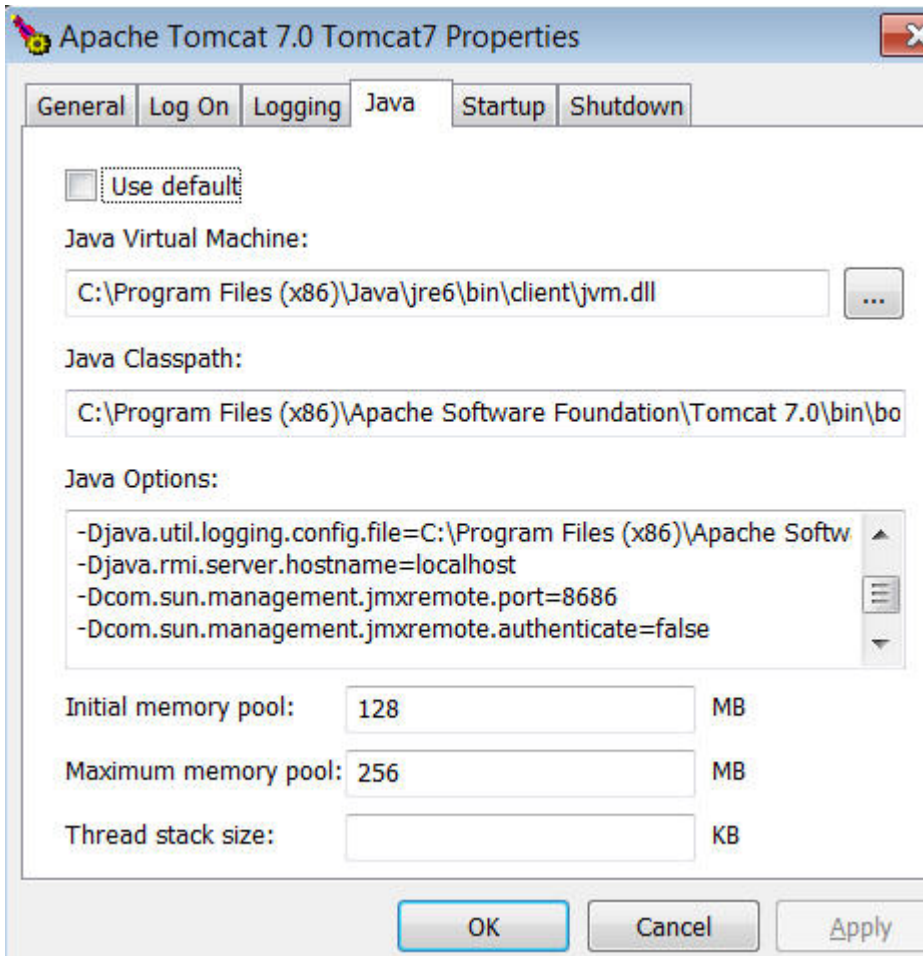
What to do next

For more information about the optimization of Worklight Server, especially the tuning of the JVM memory allocation, see "Optimization and tuning of Worklight Server" on page 181.

Configuring WebSphere Application Server and WebSphere Application Server Network Deployment:

You must configure a secure JMX connection for WebSphere Application Server and WebSphere Application Server Network Deployment.

Procedure

- IBM Worklight Foundation requires access to the SOAP port or the RMI port to perform JMX operations. By default, the SOAP port is active on a WebSphere Application Server. IBM Worklight Foundation uses the SOAP port by default, and does not run if both the SOAP and RMI ports are deactivated.
- You must activate Administrative and Application Security.

What to do next

For more information about the optimization of Worklight Server, especially the tuning of the JVM memory allocation, see "Optimization and tuning of Worklight Server" on page 181.

Installing Worklight Server administration with the Server Configuration Tool

You can use the Server Configuration Tool to install and configure Worklight Server administration.

Before you begin

Verify that the user who runs the Server Configuration Tool has the privileges that are described in "File system prerequisites" on page 72.

About this task

Important: The Server Configuration Tool does not support server farms. Therefore, you cannot use this tool to install, upgrade, or configure a server farm.

Procedure

1. Start the Worklight Server Configuration Tool.
   - On Linux: In the desktop menu, click **IBM Worklight Server** > **Server Configuration Tool**.
   - On Windows: In the **Start** menu, click **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.
2. Select **Create a Worklight Server Configuration**.
3. Name your configuration.
4. In the Configuration Description window:
   a. Enter the context root of the Worklight Administration REST service. The context root is used to create the URL to the Worklight REST Administration service. This URL is typically in the form `<URL_TO_APPLICATION_SERVER_HTTPS_PORT>/contextroot` or `<URL_TO_APPLICATION_SERVER_HTTP_PORT>/contextroot`.
   b. Enter an `environmentId`. This ID is optional and is used to distinguish between different deployments of the Worklight Server administration components in the same application server environment, for example in the same cell of WebSphere Application Server Network Deployment.

      Important: Review carefully this environment ID. It must match the environment ID of all the runtime environments that are managed by this Worklight Server administration component. If you install or upgrade the Worklight runtime environments with separate Ant files, this is particularly important because the `environmentId` attribute must match. For a server farm, all installations must also share the same `environmentId` attribute.

      The `environmentId` is an attribute of the following Ant tasks:
      - `installworklightadmin`, `updateworklightadmin`, and `uninstallworklightadmin` Ant tasks, which are documented at "Ant tasks for installation of Worklight Console and Administration Services" on page 120.

- configureapplicationserver, updateapplicationserver, unconfigureapplicationserver Ant tasks, which are documented at "Ant tasks for installation of Worklight runtime environments" on page 1146.

5. In the Console Settings window, enter the context root of the Worklight Console.

6. In the Database Properties window:

   a. Select your database type: IBM DB2, MySQL, or Oracle.

   b. In the next window, enter the details to connect to the database instance.

   c. In the Database Additional properties window, enter the parameters to connect to the administration database.

   d. If the database administrator did not create the databases in step "Optional creation of the administration database" on page 104, enter database administration credentials in the **database creation request** window.

   Note: For IBM DB2 and for Oracle, you must have an SSH access to the host where the database management system (DBMS) is installed.

   The Server Configuration Tool creates the database for you.

7. In the Application Server Choice window:

   a. Select your application server type: WebSphere Application Server, WebSphere Application Server Liberty profile, or Apache Tomcat.

   b. In the Application Server window, enter the data so that you can deploy IBM Worklight Foundation to that application server.

   c. If the application server is WebSphere Application Server Liberty profile, or Apache Tomcat, select **Create a default user** if you want to declare a user who can log in to the console as administrator to the Worklight Console.

   If the Application Server is WebSphere Application Server, select **Declare the WebSphere Administrator as a Worklight Administrator** if you want to allow the WebSphere administrator to log in to the Worklight Console.

   For more information about further configuration of security roles, see "Configuring user authentication for Worklight Server administration" on page 151.

8. When all the data is entered, click **Deploy**.

   The following effects take place.

   a. If the database administrator did not complete step "Optional creation of the administration database" on page 104, the database for the Worklight Server administration is created and access rights are granted to the user that is specified in the **database additional properties** window.

   b. If the tables for Worklight administration do not exist in the database, they are created.

   c. The Worklight administration components are installed in the application server and are connected to the database.

9. Restart the application server

10. If you are in an environment where you must protect the password of the user who can log in to the console as administrator to the Worklight Console, follow the steps in "Securing the Worklight Server administration" on page 190.

11. Open the console.

If the context root of the console was not changed in the Console Settings window, you find it at *<URL_TO_APPLICATION_SERVER_HTTPS_PORT>*/ `worklightconsole`, or if HTTPS is not supported in your application server, at the insecure URL *<URL_TO_APPLICATION_SERVER_HTTP_PORT>*/`worklightconsole`.

What to do next

Install a Worklight runtime environment. For more information, see "Deploying, updating, or undeploying Worklight Server by using the Server Configuration Tool" on page 1106.

Using Ant tasks to install Worklight Server administration

Learn about the Ant tasks that you can use to install Worklight Server administration.

Creating and configuring the database for Worklight Server administration with Ant tasks:

Use Ant tasks to create and configure the Worklight database as detailed here.

An Ant task is supplied that ensures that a WorklightAdmin 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 Foundation.

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 then connects 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 then connects 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 "Sample configuration files" on page 1156. To create a database with an example file, use a file named `create-database-`*<xxx>*`.xml`, as described in table 1 of "Sample configuration files" on page 1156, and follow the instructions in that page. If you previously performed the "Optional

creation of the administration database" on page 104, a file that is named
configure-*<xxx>*-*<yyy>*.xml, and does not require database administrator
credentials, is enough. The Ant target to run to create the database is admdatabases
(see step 6 in "Sample configuration files" on page 1156).

Prerequisite steps

Before you do 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 1110 must be completed.

To create an Apache Derby database

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 use an entire database, for example to implement the situation that
is described in "Setting up your Apache Derby databases manually" on page 1127,
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="WorklightAdmin">
      <derby database="WLADMIN" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>

  </target>
</project>
```

If you want to use just a schema in a database, in which the other schemas are
used for other purposes:

```
<?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="WorklightAdmin">
```

```
      <derby database="WorklightAdminProduction" datadir="/var/databases/derby" schema="WL62"/>
    </configuredatabase>

  </target>
</project>
```

To create a DB2 database

If you want to use an entire database, for example to implement the situation that is described in "Setting up your DB2 databases manually" on page 1121, 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="WorklightAdmin">
      <db2 database="WLADMIN" 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 use just a schema in a database, in which the other schemas are used for other purposes:

```
<?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="WorklightAdmin">
      <db2 database="PROD" server="proddb.example.com"
          user="wl6admin" password="wl6pass" schema="Admin">
        <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 MySQL database

With MySQL, the configuration looks similar to the following example:

```xml
<?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="WorklightAdmin">
      <mysql database="WLADMIN" 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 an Oracle database

You might want to use a user and schema in a database, in which the other
schemas are used for other purposes, for example to implement the situation that
is described in "Setting up your Oracle databases manually" on page 1136. In this
case, the configuration can look like the following example:

```xml
<?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="WorklightAdmin">
      <oracle database="ORCL" server="proddb.example.com"
              user="WLADMIN" 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 use an entire database, the configuration can look like the following
example:

```xml
<?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="WorklightAdmin">
            <oracle database="WLADMIN" 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>
```

Deploying the Worklight Console and Administration Services with Ant tasks:

Use Ant tasks to deploy the Worklight Console and Administration Services to an application server, and configure data sources, properties, and database drivers for use by IBM Worklight Foundation as detailed here.

Prerequisites

Verify that the user who runs the Ant tasks has the privileges that are described in "File system prerequisites" on page 72.

The `<installworklightadmin>` Ant task

You can use the `<installworklightadmin>` Ant task to deploy the Worklight Console and the Administration Services to an application server. The task:

- Installs the Administration Services as an application in the application server.
- Installs the Worklight Console as an application in the application server.
- Installs a first **worklightadmin** security role for the Administration Services and the Worklight Console.
- Configures the application server to enable JMX, if required.
- Configures the data sources for the Worklight Server administration.
- Configures Worklight configuration properties (through JNDI).
- Installs the database drivers in the application server.

To start the Ant task, you need an Ant XML file with one or more invocations of the `<installworklightadmin>` task. If you want to start the Ant task from a computer on which Worklight Server is not installed, you must copy the following files to this computer:

- *<WorklightInstallDir>*/WorklightServer/worklight-ant-deployer.jar
- *<WorklightInstallDir>*/WorklightServer/worklightadmin.war
- *<WorklightInstallDir>*/WorklightServer/worklightconsole.war

Before starting the `<installworklightadmin>` task, you must create and configure the databases. See "Creating and configuring the databases with Ant tasks" on page 1114 for details.

Note: The `<database>` element that is passed to `<installworklightadmin>` must be consistent with the `<configuredatabase>` invocation that was used when configuring the database.

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 `<uninstallworklightadmin>` Ant task

Use the `<uninstallworklightadmin>` Ant task to remove the configuration that was done by an earlier `<installworklightadmin>` invocation. This Ant task takes the same parameters as the `<installworklightadmin>` invocation.

The `<updateworklightadmin>` Ant task

You can use the `<updateworklightadmin>` Ant task to update the WAR files, following an earlier `<installworklightadmin>` invocation. It does not change the other elements of the application server configuration (such as the JMX configuration, data sources or JNDI properties).

This Ant task is useful when you upgrade to a new Worklight Server fix pack, which is an upgrade in which only the fourth digit of the Worklight Server version changed.

The `<updateworklightadmin>` Ant task takes the same parameters as the `<installworklightadmin>` invocation, except that the `<database>` element might be omitted.

Sample Ant XML files are presented in "Sample configuration files" on page 1156.

To deploy the Worklight Console and Administration Services, use a file named `configure-<xxx>-<xxx>.xml`, as described in table 1 of "Sample configuration files" on page 1156, and follow the instructions in that page. The Ant target that you run to install the Worklight Console is `adminstall` (see step 7 of "Sample configuration files" on page 1156).

Sample Ant XML file for WebSphere Application Server Liberty profile with Derby database

Note: The Apache Derby database is provided in the 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="admdatabases">
    <configuredatabase kind="WorklightAdmin">
      <derby database="WLADMIN" datadir="/var/ibm/Worklight/derby"/>
    </configuredatabase>
  </target>
```

```
<!-- Start of the adminstall target Generated by Worklight -->
<target name="adminstall">
  <installworklightadmin>
    <console install="${wladmin.console.install}"/>
    <jmx/>
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
                                  profile="${appserver.was.profile}">
        <server name="${appserver.was85liberty.serverInstance}"/>
      </websphereapplicationserver>
    </applicationserver>
    <user name="${wladmin.default.user}" role="worklightadmin" password="${wladmin.default.user.
    <database kind="WorklightAdmin">
      <derby database="${database.derby.wladmin.dbname}" datadir="${database.derby.datadir}"/>
    </database>
  </installworklightadmin>
</target>
<!-- End of the adminstall target generated by Worklight -->

<!-- Start of the minimal-admupdate target generated by Worklight -->
<target name="minimal-admupdate">
  <updateworklightadmin>
    <console install="${wladmin.console.install}"/>
    <jmx/>
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
                                  profile="${appserver.was.profile}">
        <server name="${appserver.was85liberty.serverInstance}"/>
      </websphereapplicationserver>
    </applicationserver>
    <user name="${wladmin.default.user}" role="worklightadmin" password="${wladmin.default.user.
    <database kind="WorklightAdmin">
      <derby database="${database.derby.wladmin.dbname}" datadir="${database.derby.datadir}"/>
    </database>
  </updateworklightadmin>
</target>
<!-- End of the minimal-admupdate target generated by Worklight -->

<!-- Start of the admuninstall target generated by Worklight -->
<target name="admuninstall">
  <uninstallworklightadmin>
    <console install="${wladmin.console.install}"/>
    <jmx/>
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
                                  profile="${appserver.was.profile}">
        <server name="${appserver.was85liberty.serverInstance}"/>
      </websphereapplicationserver>
    </applicationserver>
    <user name="${wladmin.default.user}" role="worklightadmin" password="${wladmin.default.user.
    <database kind="WorklightAdmin">
      <derby database="${database.derby.wladmin.dbname}" datadir="${database.derby.datadir}"/>
    </database>
  </uninstallworklightadmin>
</target>
<!-- End of the admuninstall target Generated by Worklight -->
</project>
```

Related topics

Other examples for many configurations can be found in *<worklightInstallDir>*/
WorklightServer/configuration-samples. See also:

- "Ant tasks for installation of Worklight runtime environments" on page 1146

- "Ant tasks for installation of Worklight Console and Administration Services." Reference information for the `<installworklightadmin>`, `<updateworklightadmin>`, and `<uninstallworklightadmin>` Ant tasks.
- "Sample configuration files" on page 1156. IBM Worklight Foundation includes a number of sample configuration files to help you get started with the Ant tasks.
- "Configuring multiple Worklight projects in a single environment" on page 1161. The Ant tasks that are used to configure multiple projects on a single server are documented in this section.
- "Configuring WebSphere Application Server Network Deployment servers" on page 1162. Specific considerations when configuring WebSphere Application Server Network Deployment servers through Ant tasks are documented in this section.

Ant tasks for installation of Worklight Console and Administration Services:

The `installworklightadmin`, `updateworklightadmin`, and `uninstallworklightadmin` Ant tasks are provided for the installation of the Worklight Console and Administration Services.

Task effects

`<installworklightadmin>`

> The `<installworklightadmin>` task configures an application server to run an Administration Services WAR file as a web application and, optionally, to install the Worklight Console. This task has the following effects:
> - It declares the Administration Services web application in the specified context root, by default `/worklightadmin`.
> - It declares data sources and – on WebSphere Application Server Full Profile – JDBC providers for Administration Services.
> - It deploys the Administration Services on the application server.
> - Optionally, it declares the Worklight Console as a web application in the specified context root, by default `/worklightconsole`. If the Worklight Console instance is specified, the Ant task declares the appropriate JNDI environment entry to communicate with the corresponding management service. For example:
>
> ```
> <target name="adminstall">
>   <installworklightadmin servicewar="${worklight.service.war.file}">
>     <console install="${wladmin.console.install}" warFile="${worklight.console.war.file}"/
> ```
> - Optionally, it deploys the Worklight Console WAR file on the application server.
> - It configures configuration properties for the Administration Services by using JNDI environment entries. These JNDI environment entries also give some additional information about the application server topology, for example whether the topology is a stand-alone configuration, a cluster, or a server farm.
> - Optionally, it configures users that it maps to roles used by the Worklight Console and Administration Services web applications.
> - It configures the application server for use of JMX.
> - On WebSphere Application Server, it configures the necessary custom property for the web container.

`<updateworklightadmin>`

The <updateworklightadmin> task updates an already-configured Worklight web application on an application server. This task has the following effects:

- It updates the Administration Services WAR file. This file must have the same base name as the corresponding WAR file that was previously deployed.
- It updates the Worklight Console WAR file. This file must have the same base name as the corresponding WAR file that was previously deployed.

The task does not change the application server configuration, that is, the web application configuration, data sources, JNDI environment entries, user-to-role mappings, and JMX configuration.

<uninstallworklightadmin>

The <uninstallworklightadmin> Ant task undoes the effects of an earlier run of <installworklightadmin>. This task has the following effects:

- It removes the configuration of the Administration Services web application with the specified context root. As a consequence, the task also removes the settings that were added manually to that application.
- It removes the Administration Services WAR file and the Worklight Console WAR file from the application server as an option.
- It removes the data sources and – on WebSphere Application Server Full Profile – the JDBC providers for Administration Services.
- It removes the database drivers that were used by Administration Services from the application server.
- It removes the associated JNDI environment entries.
- It removes the users configured by the installworklightadmin invocation.
- It removes the JMX configuration.

Attributes and elements

The <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> tasks have the following attributes:

Table 25. Attributes for the <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> Ant tasks

Attribute	Description	Required	Default
contextroot	Common prefix for URLs to admin services, to get information about Worklight runtime environments, applications, and adapters	No	/worklightadmin
id	Distinguishes different deployments	No	Empty
environmentId	Distinguishes different Worklight environments	No	Empty
servicewar	The WAR file for the Administration Services	No	The worklightadmin.war file is in the same directory as the worklight-ant-deployer.jar file.
shortcutsDir	Directory where to place shortcuts	No	None

Table 25. Attributes for the <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> Ant tasks (continued)

Attribute	Description	Required	Default
`wasStartingWeight`	Start order for WebSphere Application Server. Lower values start first.	No	1

`contextroot` and `id`

    The `contextroot` and `id` attributes distinguish different deployments of Worklight Console and Administration Services.

    In WebSphere Application Server Liberty profiles 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. Without this `id` attribute, two WAR files with the same context roots might conflict and these files would not be deployed.

`environmentId`

    Use the `environmentId` attribute to distinguish several environments, consisting each of Worklight Server administration and Worklight runtime web applications, that must operate independently. For example, with this option you can host a test environment, a pre-production environment, and a production environment on the same server or in the same WebSphere Application Server Network Deployment cell. This `environmentId` attribute creates a suffix that is added to MBean names that the Administration Services and the Worklight runtime projects use when they communicate through Java Management Extensions (JMX).

`servicewar`

    Use the `servicewar` attribute to specify a different directory for the Administration Services WAR file. You can specify the name of this WAR file with an absolute path or a relative path.

`shortcutsDir`

    The `shortcutsDir` attribute specifies where to place shortcuts to the Worklight Console. If you set this attribute, you can add the following files to that directory:

- `worklight-console.url`: This file is a Windows shortcut. It opens the Worklight Console in a browser.
- `worklight-console.sh`: This file is a UNIX shell script and opens the Worklight Console in a browser.
- `worklight-admin.url`: This file is a Windows shortcut. It opens in a browser and calls a REST service that returns a list of the Worklight projects that can be managed in JSON format. For each listed Worklight project, some details are also available about their artifacts, such as the number of applications, the number of adapters, the number of active devices, the number of decommissioned devices. The list also indicates whether the Worklight project runtime is running or idle.
- `worklight-admin.sh`: This file is a UNIX shell script that provides the same output as the `worklight-admin.url` file.

`wasStartingWeight`

    Use the `wasStartingWeight` attribute to specify a value that is used in WebSphere Application Server as a weight to ensure that a start order is respected. As a result of the start order value, the Administration Services web application is deployed and started before any other Worklight

runtime projects. If Worklight projects are deployed or started before the web application, the JMX communication is not established and the runtime cannot synchronize with the administration database and cannot handle server requests.

The <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> tasks support the following elements:

Table 26. Inner elements for the <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> Ant tasks

Element	Description	Count
`applicationserver`	Application server	1
`console`	Administration console	0..1
`database`	Databases	1
`jmx`	Enable Java Management Extensions	1
`property`	Properties	0..∞
`user`	User to be mapped to a security role	0..∞

To specify a Worklight Console

The <console> element collects information to customize the installation of the Worklight Console. This element has the following attributes:

Table 27. Attributes of the <console> element

Attribute	Description	Required	Default
`contextroot`	URI of the Worklight Console	No	`/worklightconsole`
`install`	Indicates whether the Worklight Console must be installed	No	Yes
`warfile`	Console WAR file	No	The `worklightconsole.war` file is in the same directory as the `worklight-ant-deployer.jar` file.

The <console> element supports the following element:

Table 28. Inner element for the <console> element

Element	Description	Count
`property`	Properties	0..∞

The <property> element specifies a deployment property to be defined in the application server. It has the following attributes:

Table 29. Attributes for the <property> element

Attribute	Description	Required	Default value
`name`	Name of the property	Yes	None

Table 29. Attributes for the <property> element (continued)

Attribute	Description	Required	Default value
`value`	Value of the property	Yes	None

By using this element, you can define your own JNDI properties or override the default value of the JNDI properties that are provided by the Administration Services and the Worklight Console WAR files.

For more information about the JNDI properties, see "List of JNDI properties for Worklight Server administration" on page 158.

To specify an application server

Use the <applicationserver> element to define the parameters that depend on the underlying application server. The <applicationserver> element supports the following elements. The attributes and inner elements of these elements are described in tables 6 through 13 of "Ant tasks for installation of Worklight runtime environments" on page 1146.

Table 30. Inner elements of the <applicationserver> element

Attribute	Description	Count
`websphereapplicationserver` or `was`	The parameters for WebSphere Application Server.	0..1
`tomcat`	The parameters for Apache Tomcat.	0..1

To specify JMX communication between the Worklight Server administration and the Worklight projects

Use the <jmx> element to ensure that a JMX connection can be established between the Worklight Server administration and the Worklight runtime projects. The <jmx> element has the following attributes, which depend on the underlying application server.

Table 31. Attributes of the <jmx> element

Attribute	Description	Required	Default
`libertyAdminUser`	The administrator (for Liberty only)	No	None
`libertyAdminPassword`	The administrator password (for Liberty only).	No	None
`CreateLibertyAdmin`	Whether the **admin** user must be created in the basic registry, if it does not exist (for Liberty only).	No	true
`tomcatRMIPort`	The RMI port that Apache Tomcat uses to connect to Worklight projects (for Tomcat only)	No	8686
`tomcatSetEnvConfig`	Prevents automatic modification of `setenv.bat` and `setenv.sh` scripts. The valid values are `manual` and `auto`.	No	auto

Note: The **libertyAdminUser** and **libertyAdminPassword** attributes are not mandatory, but if you define one of these attributes, you must also define the other.

libertyAdminUser
libertyAdminCreate
libertyAdminPassword

> You use these attributes to create an `admin` user in the `server.xml` file, which is the configuration file for Liberty, in the basic registry section.

tomcatRMIPort

> If the default port 8686 is not available on the system, you use this attribute to specify a different port for JMX communication between the Worklight Server administration and the managed Worklight projects. In this case, the port values range from 1 to 65535.

tomcatSetEnvConfig

> You use this attribute to allow or prevent the `<installworklightadmin>` and `<uninstallworklightadmin>` Ant tasks from adding or removing contents to the `setenv.sh` or `setenv.bat` script, in the `<TomcatRootInstallDir>`/bin directory.
>
> **Important:** Security warning. The default value `auto` does not secure the JMX communication. This setting is not suitable for production environments. In production environments, you must manually configure JMX with authentication, as described in the Enabling JMX Remote page of the Apache Tomcat user documentation.
> Use the following values for this attribute:
>
> - `manual`: The `<installworklightadmin>` and `<uninstallworklightadmin>` Ant tasks do not update the `setenv.bat` and `setenv.sh` script for JMX usage.
>
>   If you select the value `manual`, you must update the scripts manually to define the RMI port that is used for JMX communications internally between the Administration Services and the Worklight runtime environment, whether this connection must be secured or not with user or role authentication, or SSL. For more information, see the documentation of the JVM that you are using.
>
> - `auto`: The `<installworklightadmin>` and `<uninstallworklightadmin>` Ant tasks update the `setenv.bat` and `setenv.sh` script automatically, for JMX usage. If these scripts do not exist, they are created before they are updated.
>
>   If you select the `auto` value, the following modifications are made to extend the CATALINA_OPTS environment variable:
>
>   - For `setenv.bat`:
>
>     ```
>     REM Allow to inspect the MBeans through jconsole
>     set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote
>
>     REM Configure JMX.
>     set CATALINA_OPTS=%CATALINA_OPTS% -Djava.rmi.server.hostname=localhost
>     set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.port=8686
>     set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.authenticate=false
>     set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.ssl=false
>     ```
>
>   - For `setenv.sh`:

```
                    # Allow to inspect the MBeans through jconsole
                    CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote"

                    # Configure JMX.
                    CATALINA_OPTS="$CATALINA_OPTS -Djava.rmi.server.hostname=localhost"
                    CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.port=8686"
                    CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
                    CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.ssl=false"
```

To specify a connection to the administration database

The <database> element collects the parameters that specify a data source
declaration in an application server to access the administration database.

You must declare a single database: <database kind="WorklightAdmin">. You
specify the <database> element similarly to the <configuredatabase> Ant task,
except that the <database> element does not have the <dba> and <client>
elements. It might have <property> elements.

The <database> element has the following attributes:

Table 32. Attributes of the <database> element

Attribute	Description	Required	Default
kind	The kind of database (**WorklightAdmin**)	Yes	None
validate	To validate whether the database is accessible	No	True

The <database> element supports the following elements. For more information
about the configuration of these database elements, see 18 through 28 in "Ant tasks
for installation of Worklight runtime environments" on page 1146.

Table 33. Inner elements for the <applicationserver> element

Element	Description	Count
db2	Parameter for DB2 databases	0..1
derby	Parameter for Apache Derby databases	0..1
mysql	Parameter for MySQL databases	0..1
oracle	Parameter for Oracle databases	0..1
driverclasspath	Parameter for JDBC driver class path	0..1

To specify a user and a security role

The <user> element collects the parameters about a user to include in a certain
security role for an application.

Table 34. Attributes of the <user> element

Attribute	Description	Required	Default
role	A valid security role for the application	Yes	None
name	The user name	Yes	None

Table 34. Attributes of the <user> element (continued)

Attribute	Description	Required	Default
`password`	The password if the user needs to be created	No	None

After you defined users by using the `<user>` element, you can map them to any of the following roles for authentication in the Worklight Console.

- `worklightmonitor`
- `worklightoperator`
- `worklightdeployer`
- `worklightadmin`

For information about which authorizations are implied by the specific roles, see the chapter about the "REST Services API" on page 1016.

Tip: If users exist in an external LDAP directory, set only the `role` and `name` attributes but do not define any passwords.

Manually installing Worklight Server administration

You can install the Worklight Server administration manually instead of using the Ant task or the Server Configuration Tool. You might also want to reconfigure Worklight Server so that it uses a different database or schema from the one that was specified during the first installation of Worklight Server. This reconfiguration depends on the type of database and the kind of application server.

Configuring the DB2 database manually for the Worklight Server administration:

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 Worklight Server administration" on page 104.
2. Create the tables in the database. This step is described in "Setting up your DB2 database manually for the Worklight Server administration."
3. Perform the application server-specific setup as the following list shows.

Setting up your DB2 database manually for the Worklight Server administration:

You can set up your DB2 database for the Worklight Server administration manually.

About this task

Set up your DB2 database for the Worklight Server administration 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 **WLADMIN**:
   ```
   CREATE DATABASE WLADMIN COLLATE USING SYSTEM PAGESIZE 32768
   CONNECT TO WLADMIN
   GRANT CONNECT ON DATABASE TO USER worklight
   QUIT
   ```

4. Run DB2 with the following commands to create the **WLADMIN** tables, in a schema named **WLADMSC** (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 WLADMIN
   db2 SET CURRENT SCHEMA = 'WLADMSC'
   db2 -vf <worklight_install_dir>/WorklightServer/databases/create-worklightadmin-db2.sql -t
   ```

Configuring Liberty profile for DB2 manually for Worklight Server administration:

If you want to manually set up and configure your DB2 database for Worklight Server administration 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 Worklight administration database. -->
   <dataSource jndiName="jdbc/WorklightAdminDS" transactional="false">
     <jdbcDriver libraryRef="DB2Lib"/>
     <properties.db2.jcc databaseName="WLADMIN"  currentSchema="WLADMSC"
                         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 **WLADMIN** 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 Worklight Server administration:

If you want to manually set up and configure your DB2 database for the Worklight Server administration 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/WorklightAdmin/db2.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/WorklightAdmin/db2.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/WorklightAdmin/db2.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/WorklightAdmin/db2.
   - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/WorklightAdmin/db2.

   If the directory for the JDBC driver JAR file does not exist, you must 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) to the directory that you determined in step 1.
3. Set up the JDBC provider:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Set **Database type** to **DB2**.
   e. Set **Provider type** to **DB2 Using IBM JCC Driver**.
   f. Set **Implementation Type** to **Connection pool data source**.
   g. Set **Name** to **DB2 Using IBM JCC Driver**.
   h. Click **Next**.

i. Set the **Class path** to the set of JAR files in the directory that you
      determined in step 1, one per line, replacing *WAS_INSTALL_DIR*/profiles/
      profile-name with the WebSphere Application Server variable reference
      ${USER_INSTALL_ROOT}.
   j. Do not set **Native library path**.
   k. Click **Next**.
   l. Click **Finish**.
   m. The JDBC provider is created.
   n. Click **Save**.

4. Create a data source for the administration database:
   a. Select **Resources** > **JDBC** > **Data Sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New** to create a data source.
   d. Set the **Data source name** to **administration database**.
   e. Set **JNDI Name** to **jdbc/WorklightAdminDS**.
   f. Click **Next**.
   g. Enter properties for the data source, for example:
      - **Driver type**: 4
      - **Database Name**: WLADMIN
      - **Server name**: localhost
      - **Port number**: 50000 (default)

      Leave **Use this data source in (CMP)** selected.
   h. Click **Next**.
   i. Create JAAS-J2C authentication data, specifying the DB2 user name and
      password as its properties. If necessary, go back to the data source creation
      wizard, by repeating steps 4a to 4h.
   j. Select the authentication alias that you created in the **Component-managed
      authentication alias** combination box (not in the **Container-managed
      authentication alias** combination box).
   k. Click **Next** and **Finish**.
   l. Click **Save**.
   m. In **Resources** > **JDBC** > **Data sources**, select the new data source.
   n. Click **WebSphere Application Server data source properties**.
   o. Select the **Non-transactional data source** check box.
   p. Click **OK**.
   q. Click **Save**.
   r. Click **Custom properties** for the data source, select property **currentSchema**,
      and set the value to the schema used to create the Worklight Server
      administration tables (WLADMSC in this example).

5. Test the data source connection by selecting **Data Source** and clicking **Test
   Connection**.

6. For WebSphere Application Server Network Deployment, click **System
   administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

Configuring Apache Tomcat for DB2 manually for Worklight Server administration:

If you want to manually set up and configure your DB2 database for Worklight
Server administration 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/WorklightAdminDS"
            username="worklight"
            password="password"
            type="javax.sql.DataSource"
            url="jdbc:db2://server:50000/WLADMIN:currentSchema=WLADMSC;"/>
  ...
</Context>
```

   Where **worklight** after **user=** is the name of the system user with "CONNECT" access to the **WLADMIN** 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 the Worklight Server administration:

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 the Worklight Server administration."

2. Configure the application server to use this database setup. Go to one of the following topics:

   * "Configuring Liberty profile for Derby manually for Worklight Server administration" on page 132
   * "Configuring WebSphere Application Server for Derby manually for Worklight Server administration" on page 132
   * "Configuring Apache Tomcat for Derby manually for the Worklight Server administration" on page 134

Setting up your Apache Derby database manually for the Worklight Server administration:

You can set up your Apache Derby database for the Worklight Server administration manually using the procedures in this section.

About this task

Set up your `Apache Derby` database for the Worklight Server administration 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:WLADMIN;user=WLADMIN;create=true';
   run '<worklight_install_dir>/WorklightServer/databases/create-worklightadmin-derby.sql';
   quit;
   ```

Configuring Liberty profile for Derby manually for Worklight Server administration:

If you want to manually set up and configure your Apache Derby database for Worklight Server administration 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 Worklight administration database. -->
<dataSource jndiName="jdbc/WorklightAdminDS" transactional="false" statementCacheSize="10">
  <jdbcDriver libraryRef="derbyLib"
              javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSou
  <properties.derby.embedded databaseName="DERBY_DATABASES_DIR/WLADMIN" user="WLADMIN"
                             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 Worklight Server administration:

If you want to manually set up and configure your Apache Derby database for the Worklight Server administration 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/WorklightAdmin/derby.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/WorklightAdmin/derby.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/WorklightAdmin/derby.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/WorklightAdmin/derby.
   - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/WorklightAdmin/derby.

   If the directory for the JDBC driver JAR file does not exist, you must create it.
2. Add the Derby JAR file from WORKLIGHT_INSTALL_DIR/ApplicationCenter/tools/lib/derby.jar to the directory that you determined in step 1.
3. Set up the JDBC provider.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the appropriate scope from the **Scope** combination box.
   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 that you 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 administration database.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Set **Data source Name** to **administration database**.
   e. Set **JNDI name** to jdbc/WorklightAdminDS.
   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 **administration Database** datasource that you created.
>
> m. Under **Additional Properties**, click **Custom properties**.
>
> n. Click **databaseName**.
>
> o. Set **Value** to the path to the `WLADMIN` database that is created in "Setting up your Apache Derby database manually for the Worklight Server administration" on page 131.
>
> p. Click **OK**.
>
> q. Click **Save**.
>
> r. At the top of the page, click **administration atabase**.
>
> 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 **administration Database** datasource that you created.
>
> x. Click **test connection** (only if you are not on the console of a WebSphere Application Server Deployment Manager).

5. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

Configuring Apache Tomcat for Derby manually for the Worklight Server administration:

If you want to manually set up and configure your Apache Derby database for the Worklight Server administration 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/WorklightServer/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/WorklightAdminDS"
            username="WLADMIN"
            password=""
            type="javax.sql.DataSource"
            url="jdbc:derby:DERBY_DATABASES_DIR/WLADMIN"/>
  ...
</Context>
```

Configuring the MySQL database manually for the Worklight Server administration:

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 Worklight Server administration" on page 105.
2. Create the tables in the database. This step is described in "Setting up your MySQL database manually for the Worklight Server administration."
3. Perform the application server-specific setup as the following list shows.

Setting up your MySQL database manually for the Worklight Server administration:

You can set up your MySQL database for the Worklight Server administration 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 WLADMIN CHARACTER SET utf8 COLLATE utf8_general_ci;
   GRANT ALL PRIVILEGES ON WLADMIN.* TO 'worklight'@'Worklight-host'IDENTIFIED BY 'worklight';
   GRANT ALL PRIVILEGES ON WLADMIN.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
   FLUSH PRIVILEGES;

   USE WLADMIN;
   SOURCE <worklight_install_dir>/WorklightServer/databases/create-worklightadmin-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 Foundation 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 Worklight Server administration:

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

About this task

Complete the MySQL database setup procedure before continuing.

Procedure

1. Add the MySQL JDBC driver JAR file to $LIBERTY_HOME/wlp/usr/shared/
   resources/mysql. If that directory does not exist, create it.

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

```
<!-- Declare the jar files for MySQL access through JDBC. -->
<library id="MySQLLib">
  <fileset dir="${shared.resource.dir}/mysql" includes="*.jar"/>
</library>


<!-- Declare the IBM Worklight administration database. -->
<dataSource jndiName="jdbc/WorklightAdminDS" transactional="false">
  <jdbcDriver libraryRef="MySQLLib"/>
  <properties databaseName="WLADMIN"
              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 Worklight Server
administration:*

If you want to manually set up and configure your MySQL database for Worklight
Server administration 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/WorklightAdmin/mysql.
   - For deployment to a WebSphere Application Server Network Deployment
     cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/
     WorklightAdmin/mysql.
   - For deployment to a WebSphere Application Server Network Deployment
     cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-
     name*/clusters/*cluster-name*/WorklightAdmin/mysql.
   - For deployment to a WebSphere Application Server Network Deployment
     node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-
     name*/nodes/*node-name*/WorklightAdmin/mysql.

- For deployment to a WebSphere Application Server Network Deployment
  server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/WorklightAdmin/mysql.

If the directory for the JDBC driver JAR file does not exist, you must create it.

2. Add the MySQL JDBC driver JAR file that you downloaded from Download
   Connector/J 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. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Create a **JDBC provider** named **MySQL**.
   e. Set **Database type** to **User defined**.
   f. Set **Scope** to **Cell**.
   g. Set **Implementation class** to **com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource**.
   h. Set the **Class path** to the JAR file in the directory that you determined in step 1, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.
   i. Save your changes.
4. Create a data source for the administration database:
   a. Select **Resources** > **JDBC** > **Data Sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New** to create a data source.
   d. Type any name (for example, Worklight administration Database).
   e. Set **JNDI Name** to jdbc/WorklightAdminDS.
   f. Use the existing **JDBC Provider MySQL**, defined in the previous step.
   g. Set Scope to **New**.
   h. On the **Configuration** tab, select **Non-transactional data source**.
   i. Click **Next** a number of times, leaving all other settings as defaults.
   j. 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 = WLADMIN
      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**.

7. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

Configuring Apache Tomcat for MySQL manually for Worklight Server administration:

If you want to manually set up and configure your MySQL database for Worklight Server administration 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/WorklightAdminDS"
               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/WLADMIN"/>
     ...
   </Context>
   ```

Configuring the Oracle database manually for the Worklight Server administration:

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 Worklight Server administration" on page 106.
2. Create the tables in the database. This step is described in "Setting up your Oracle database manually for the Worklight Server administration."
3. Perform the application server-specific setup as the following list shows.

Setting up your Oracle database manually for the Worklight Server administration:

You can set up your Oracle database for the Worklight Server administration 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 WLADMIN, either by using Oracle Database Control, or by using the Oracle SQLPlus command-line interpreter.

   Create the user for the 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 WLADMIN with the following attributes:

   ```
   Profile: DEFAULT
   Authentication: password
   Default tablespace: USERS
   Temporary tablespace: TEMP
   Status: UNLOCK
   Add system privilege: CREATE SESSION
   Add system privilege: CREATE SEQUENCE
   Add system privilege: CREATE TABLE
   Add system privilege: UNLIMITED TABLESPACE
   ```

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

   ```
   CONNECT system/<system_password>@ORCL
   CREATE USER WLADMIN IDENTIFIED BY password;
   GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WLADMIN;
   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 Worklight database and Worklight reports database:

   a. Using the Oracle SQLPlus command-line interpreter, create the required tables for the IBM administration database by running the create-worklightadmin-oracle.sql file:

   ```
   CONNECT WLADMIN/<WLADMIN_password>@ORCL
   @<worklight_install_dir>/WorklightServer/databases/create-worklightadmin-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 Worklight Server administration:

If you want to manually set up and configure your Oracle database for Worklight Server administration 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 Worklight administration database. -->
<dataSource jndiName="jdbc/WorklightAdminDS" transactional="false">
  <jdbcDriver libraryRef="OracleLib"/>
  <properties.oracle driverType="thin"
                     serverName="oserver" portNumber="1521"
                     databaseName="ORCL"
                     user="WLADMIN" password="WLADMIN_password"/>
</dataSource>
```

where **WLADMIN** after **user=** is the user name, **WLADMIN_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 the Worklight Server administration:

If you want to manually set up and configure your Oracle database for the Worklight Server administration 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/WorklightAdmin/oracle.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/WorklightAdmin/oracle.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/WorklightAdmin/oracle.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/WorklightAdmin/oracle.
   - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/WorklightAdmin/oracle.

   If the directory for the JDBC driver JAR file does not exist, you must 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**.
    b. Select the appropriate scope from the **Scope** combination box.
    c. Click **New**.
    d. Complete the JDBC Provider fields as indicated in the following table:

Table 35. JDBC Provider field values

Field	Value
Database type	Oracle
Provider type	Oracle JDBC Driver
Implementation type	Connection pool data source
Name	Oracle JDBC Driver

    e. Click Next.
    f. Set the **Class path** to the JAR file in the directory that you determined in step 1, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.
    g. Click **Next**.
       The JDBC provider is created.
4. Create a data source for the administration database:
    a. Click **Resources** > **JDBC** > **Data sources**.
    b. Select the appropriate scope from the **Scope** combination box.
    c. Click **New**.
    d. Set **Data source name** to **Oracle JDBC Driver DataSource**.
    e. Set **JNDI name** to jdbc/WorklightAdminDS.
    f. Click **Next**.
    g. Click **Select an existing JDBC provider** and select **Oracle JDBC driver** from the list.
    h. Click **Next**.
    i. 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).
    j. Click **Next** twice.
    k. Click **Resources** > **JDBC** > **Data sources** > **Oracle JDBC Driver DataSource** > **Custom properties**.
    l. Set **oracleLogPackageName** to **oracle.jdbc.driver**.
    m. Set **user = WLADMIN**.
    n. Set **password = *WLADMIN_password***.
    o. Click **OK** and save the changes.
    p. In **Resources** > **JDBC** > **Data sources**, select the new data source.
    q. Click **WebSphere Application Server data source properties**.
    r. Select the **Non-transactional data source** check box.
    s. Click **OK**.
    t. Click **Save**.
5. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

Configuring Apache Tomcat for Oracle manually for Worklight Server administration:

If you want to manually set up and configure your Oracle database for Worklight Server administration 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/WorklightAdminDS"
            auth="Container"
            type="javax.sql.DataSource"
            driverClassName="oracle.jdbc.driver.OracleDriver"
            url="jdbc:oracle:thin:@oserver:1521:ORCL"
            username="WLADMIN"
            password="WLADMIN_password"/>
  ...
</Context>
```

   Where **WLADMIN** after **username=** is the name of the system user with "CONNECT" access to the **WLADMIN** database that you have previously created, and **WLADMIN_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 Administration Services and Worklight Console and configuring the application server manually:

The procedure to manually deploy the Administration services and Worklight Console to an application server depends on the type of application server being configured.

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

Note: Using the Worklight Server installer to install Worklight Server administration 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 Worklight Server administration. You must deploy the `worklightconsole.war` and `worklightadmin.war` files to your Worklight Server administration. The files are located in *wl_install_dir*/WorklightServer.

Configuring WebSphere Application Server Liberty profile for Worklight Server administration manually:

To configure WebSphere Application Server Liberty profile for the Worklight Server administration manually, you must modify the `server.xml` file.

About this task

In addition to modifications for the databases that are described in "Manually installing Worklight Server administration" on page 127, 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>jndi-1.0</feature>
   <feature>restConnector-1.0</feature>
   <feature>appSecurity-1.0</feature>
   ```

2. Follow the instructions from the IBM WebSphere Application Server user documentation to configure the secure JMX connection.

3. Add the following global JNDI entries in the server.xml file:

   ```
   <jndiEntry jndiName="ibm.worklight.admin.jmx.host" value="localhost"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.port" value="9443"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.user" value="WorklightRESTUser"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.pwd" value="WorklighRESTUserPassword"/>
   <jndiEntry jndiName="ibm.worklight.topology.platform" value="Liberty"/>
   <jndiEntry jndiName="ibm.worklight.topology.clustermode" value="Standalone"/>
   ```

   Where:
   - ibm.worklight.admin.jmx.port is the HTTPS port. You can find its value in the httpEndpoint element of the server.xml file.
   - ibm.worklight.admin.jmx.user is a user with the <administrator-role> that you created in step 2.
   - ibm.worklight.admin.jmx.pwd is the password of that user.

4. Modify the web container definition with the following values:

   ```
   <webContainer invokeFlushAfterService="false" deferServletLoad="false"/>
   ```

5. Copy the following WAR files to the apps directory of the liberty server: <WorklightInstallDir>/WorklightServer/worklightadmin.war and <WorklightInstallDir>/WorklightServer/worklightconsole.war.

   Note: the apps directory is in the same directory as the server.xml file.

6. Declare the Administration Services and Worklight Console applications:

   ```
   <!-- Declare the Administration Services application. -->
   <application id="worklightadmin" name="worklightadmin" location="worklightadmin.war" type="war
     <application-bnd>
       <security-role name="worklightadmin">
       <!-- This example adds a user to the worklightadmin security-role <user name="worklightUse
       </security-role>
       <security-role name="worklightdeployer">
       </security-role>
       <security-role name="worklightmonitor">
       </security-role>
       <security-role name="worklightoperator">
       </security-role>
     </application-bnd>
     <classloader delegation="parentLast">
       <commonLibrary>
   ```

```
            <!-- Important: the version number of the following cryptographic JAR file might change acc
            <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.jar"/>
        </commonLibrary>
      </classloader>
    </application>

    <!-- Declare the Worklight Console application. -->
    <application id="worklightconsole" name="worklightconsole" location="worklightconsole.war" type="
      <application-bnd>
        <security-role name="worklightadmin">
        <!-- This example adds a user to the worklightadmin security-role <user name="worklightUser"/
        </security-role>
        <security-role name="worklightdeployer">
        </security-role>
        <security-role name="worklightmonitor">
        </security-role>
        <security-role name="worklightoperator">
        </security-role>
      </application-bnd>
    </application>

    <jndiEntry jndiName="worklightconsole/ibm.worklight.admin.endpoint" value='"*://*:*/worklightadmi
```

Note: For more information about how to configure a user registry for Liberty
profile, see Configuring a user registry for the Liberty profile.

The JNDI property `worklightconsole/ibm.worklight.admin.endpoint` is
prefixed by the context root of the Worklight Console application, in this
example `worklightconsole`. The value of this property is the end point to the
Worklight administration.

The syntax `"*://*:*/worklightadmin"` means that the same URL that is used to
contact the Worklight Console is used, but the context root of the Worklight
Console is replaced by `worklightadmin`.

You might also specify the full endpoint, for example: `http://`
`myhostname.mydomain.com:9080/worklightadmin`.

7. If the database is Oracle, add the `commonLibraryRef` attribute to the class loader
   of the `worklightadmin` application.

```
...
<classloader delegation="parentLast"  commonLibraryRef="OracleLib">
  <commonLibrary>
...
```

The name of the library reference (`OracleLib` in this example) must be the ID of
the library that contains the JDBC JAR file, which is declared in the procedure
that is documented in "Configuring Liberty profile for Oracle manually for
Worklight Server administration" on page 139.

Configuring WebSphere Application Server for Worklight Server administration
manually:

To configure WebSphere Application Server for Worklight Server administration
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 that is
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 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 be enabled only if administrative security is enabled.

   c. Ensure that **Enable application security** is selected.

   d. Click **OK**.

   e. Save the changes.

   For more information, see Enabling security in WebSphere Application Server user documentation.

3. Review the server class loader policy: Click **Servers** > **Server Types** > **WebSphere application servers**, and select the server used for 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**.

4. Create the Worklight Server administration JDBC data source and provider. See the instructions in the appropriate subsection in "Manually installing Worklight Server administration" on page 127.

5. If you install on WebSphere Application Server Network Deployment, find the SOAP port of the deployment manager by clicking System Administration/Deployment manager.

   a. In **Additional properties**, open **Ports**.

   b. Take note of the value `SOAP_CONNECTOR_ADDRESS`, because you need it to set the value of the `ibm.worklight.admin.jmx.dmgr.port` environment entry for the Administration Services.

6. Install the Administration Services 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. Go to the Worklight Server installation directory `WL_INSTALL_DIR/WorklightServer`.

   c. Select **worklightadmin.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 resource references to resources page, and enter the JNDI name of the data source that you created in step 4.

   g. Click **Next** until you reach the Map context roots for web modules page.

   h. In the **Context Root** field, type `/worklightadmin`.

   i. Click **Next**.

j. In Map environment entries for web modules:

- If you install by using the Deployment Manager in the WebSphere Application Server Network Deployment product, enter the following values:
  - For the entry `ibm.worklight.admin.jmx.dmgr.host`, enter the host name of the deployment manager.
  - For the entry `ibm.worklight.admin.jmx.dmgr.port`, enter the SOAP port of the deployment manager that you noted in step 5.b.
  - For the entry `ibm.worklight.topology.platform`, enter `WAS`.
  - For the entry `ibm.worklight.topology.clustermode`, enter `Cluster`.
- If you install on a stand-alone server:
  - For the entry `ibm.worklight.topology.platform`, enter `WAS`.
  - For the entry `ibm.worklight.topology.clustermode`, enter `Standalone`.

k. Click **Next** until you reach the last step, and click **Finish**.

l. Click **Save**.

7. Configure the class loader policies for the Administration Services and then start the application:

a. Click the **Manage Applications** link, or click **Applications** > **Applications Types** > **WebSphere enterprise applications**.

b. From the list of applications, click **worklightadmin_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 **Detail Properties** section, click the **Startup behavior** link.

g. In **Startup Order**, enter 1, and click **OK**.

h. In the **Modules** section, click **Manage Modules**.

i. From the list of modules, click the **Worklight Administration Services** module.

j. In the **Class loader order** pane, click **Classes loaded with local class loader first (parent last)**.

k. Click **OK** twice.

l. Click **Save**.

m. Select **worklightadmin_war** and click **Start**.

8. Install the Worklight 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. Go to the Worklight Server installation directory `WL_INSTALL_DIR/WorklightServer`.

c. Select **worklightconsole.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 modules page.

g. In the **Context Root** field, type /worklightconsole.

h. Click **Next**.

i. In Map environment entries for web modules, enter the value
   ://:*/worklightadmin for the entry ibm.worklight.admin.endpoint.

j. Click **Next** until you reach the last step, and click **Finish**.

k. Click **Save**.

9. Configure the class loader policies for the Worklight Console and start the
   application:

   a. Click the **Manage Applications** link, or click **Applications** > **Application
      Types** > **WebSphere enterprise applications**.

   b. From the list of applications, click **worklightconsole_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 **Detail Properties** section, click the **Startup behavior** link.

   g. In **Startup Order**, enter 1, and click **OK**.

   h. In the **Modules** section, click **Manage Modules**.

   i. From the list of modules, click the **Worklight Console** module.

   j. In the **Class loader order** pane, click **Classes loaded with local class loader
      first (parent last)**.

   k. Click **OK** twice.

   l. Click **Save**.

   m. Click **Applications** > **Application Types** > **WebSphere enterprise
      applications**.

   n. Select **Select** for **worklightconsole_war** and click **Start**.

Results

You can now access the Worklight Server administration at http://
<server>:*<port>*/worklightconsole, where *server* is the host name of your server
and *port* is the port number (by default 9080).

What to do next

For more steps to configure Worklight Server administration, see "Configuring
WebSphere Application Server full profile for Worklight Server administration" on
page 151.

Configuring Apache Tomcat for Worklight Server administration manually:

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

Before you begin

The following prerequisites are required:

- The configuration of database for Worklight Server administration must be performed. For more information about various databases configuration, see "Manually installing Worklight Server administration" on page 127.
- The CATALINA_OPTS options to enable JMX must be defined as described in "Configuring Apache Tomcat" on page 108.

Procedure

1. 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 Worklight Console and Administration Services applications and a user registry:

```
<!-- Declare the Administration Services application. -->
<Context docBase="worklightadmin" path="/worklightadmin">

  <!-- Declare the JNDI environment entries for the Administration Services. -->
  <Environment name="ibm.worklight.topology.platform" value="Tomcat" type="java.lang.String" 
  <Environment name="ibm.worklight.topology.clusterMode" value="Standalone" type="java.lang.St

  <!-- Declare the administration database. -->
  <!-- <Resource name="jdbc/WorklightAdminDS" type="javax.sql.DataSource" ... /> -->

</Context>

<!-- Declare the Worklight Console application. -->
<Context docBase="worklightconsole" path="/worklightconsole">

  <!-- Declare the JNDI environment entries for the Worklight Console. -->
  <Environment name="ibm.worklight.admin.endpoint" value="*://*:*/worklightadmin" type="java.
</Context>

<!-- Declare the user registry for the Worklight Server administration.
     The MemoryRealm recognizes the users defined in conf/tomcat-users.xml.
     For other choices, see Apache Tomcat "Realm Configuration HOW-TO"
     http://tomcat.apache.org/tomcat-7.0-doc/realm-howto.html . -->
<Realm className="org.apache.catalina.realm.MemoryRealm"/>
```

       where you must uncomment and fill in the `<Resource>` element to declare the administration database as described in one of the sections:

       - "Configuring Apache Tomcat for DB2 manually for Worklight Server administration" on page 130
       - "Configuring Apache Tomcat for Derby manually for the Worklight Server administration" on page 134
       - "Configuring Apache Tomcat for MySQL manually for Worklight Server administration" on page 138
       - "Configuring Apache Tomcat for Oracle manually for Worklight Server administration" on page 142

2. Copy the Worklight Server administration WAR files to Tomcat.

    - On UNIX and Linux systems: cp *WL_INSTALL_DIR*/WorklightServer/*.war *TOMCAT_HOME*/webapps
    - On Windows systems:

```
copy /B WL_INSTALL_DIR\WorklightServer\worklightconsole.war TOMCAT_HOME\webapps\worklightconso
copy /B WL_INSTALL_DIR\WorklightServer\worklightadmin.war TOMCAT_HOME\webapps\worklightadmin.wa
```

3. Start Tomcat.

What to do next

For additional steps to configure the Worklight Server administration, see "Configuring Apache Tomcat for Worklight Server administration" on page 153.

Defining a server farm for Worklight Server administration:

You configure IBM Worklight Foundation to work in a server farm topology by using JNDI properties and by defining the farm nodes correctly in XML.

To define a server farm in IBM Worklight Foundation, you must define the farm nodes in a flat XML file. A server farm works with a homogeneous list of farm node types; that is, all the member nodes must use the same type of application server. The following application servers are supported:
- WebSphere Application Server full profile
- WebSphere Application Server Liberty profile
- Apache Tomcat

If all the nodes do not use the same type of application server, the server farm is not guaranteed to work correctly.

The following JNDI properties are mandatory because they define the type and location of the server farm: **ibm.worklight.farm.type** and **ibm.worklight.farm.definition.location**

The **ibm.worklight.farm.type** property accepts only the FILE value. This value enables the list of farm nodes to be built from an XML file where the farm node properties are defined. Use the **ibm.worklight.farm.definition.location** property to define the full path of the XML file.

To be readable, the XML file must comply with the *wl_install_dir*/WorklightServer/FarmSchema.xsd schema definition. You can validate the structure of your customized flat file against the schema.

Schema definition of the flat file that defines server farm nodes

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Licensed Materials - Property of IBM
 5725-I43 (C) Copyright IBM Corp. 2006, 2014. All Rights Reserved.
 US Government Users Restricted Rights - Use, duplication or
        disclosure restricted by GSA ADP Schedule Contract with IBM Corp. -->

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >

  <xs:simpleType name="PortNumberType">
  <xs:restriction base="xs:integer">
   <xs:minInclusive value="0" />
   <xs:maxInclusive value="65535" />
   <xs:whiteSpace value="collapse"></xs:whiteSpace>
  </xs:restriction>
 </xs:simpleType>

  <xs:attribute name="Hostname" type="xs:string" />
 <xs:attribute name="JMXPortNumber" type="PortNumberType" />
 <xs:attribute name="TomcatPortNumber" type="PortNumberType" />

 <xs:complexType name="TomcatNodeType">
  <xs:attribute ref="Hostname" use="required" />
  <xs:attribute name="ServerID" type="xs:string" use="required"  />
  <xs:attribute ref="TomcatPortNumber" use="optional" />
```

```
   <xs:attribute ref="JMXPortNumber" use="required"  />
 </xs:complexType>

 <xs:complexType name="LibertyNodeType">
  <xs:attribute ref="Hostname" use="required" />
  <xs:attribute name="ServerID" type="xs:string" use="required"  />
  <xs:attribute ref="JMXPortNumber" use="required"   />
  <xs:attribute name="AdminUser" type="xs:string" use="required"   />
  <xs:attribute name="AdminPass" type="xs:string" use="required" />
 </xs:complexType>


  <xs:element name="TomcatNode" type="TomcatNodeType"/>
 <xs:element name="LibertyNode" type="LibertyNodeType"/>
 <xs:element name="WasNode" type="LibertyNodeType"/>

 <xs:element name="Farm">
  <xs:complexType>
   <xs:sequence>
    <xs:choice maxOccurs="unbounded" minOccurs="1" >
     <xs:element ref="TomcatNode" />
     <xs:element ref="LibertyNode" />
     <xs:element ref="WasNode" />
    </xs:choice>
   </xs:sequence>
  </xs:complexType>
 </xs:element>

</xs:schema>
```

Example of server farm nodes for WebSphere Application Server full profile

The following example shows how to configure the nodes in a server farm that run
on WebSphere Application Server configured as full profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="FarmSchema
 <WasNode Hostname="someWasHost1" ServerID ="wasServerID1" AdminPass="wasPassword1" AdminUser="wasAdm
 <WasNode Hostname="someWasHost2" ServerID ="wasServerID2" AdminPass="wasPassword2" AdminUser="wasAdm
</Farm>
```

Example of server farm nodes for the Liberty profile

The following example shows how to configure the nodes in a server farm on
application servers that run the Liberty profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="FarmSchema

 <LibertyNode Hostname="someLibertyHost1" ServerID ="libertyServerID1" AdminPass="libertyPassword1" /
 <LibertyNode Hostname="someLibertyHost2" ServerID ="libertyServerID2" AdminPass="libertyPassword2" /
 <LibertyNode Hostname="someLibertyHost3" ServerID ="libertyServerID3" AdminPass="libertyPassword3" /

</Farm>
```

Example of server farm nodes for Apache Tomcat

The following example shows how to configure the nodes in a server farm that run
on Apache Tomcat application servers.

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="FarmSchema
 <TomcatNode ServerID ="tomcatServerID1" Hostname="someTomcatHost1" JMXPortNumber="12345" TomcatPort
 <TomcatNode ServerID ="tomcatServerID2" Hostname="someTomcatHost2" JMXPortNumber="12345" TomcatPort
</Farm>
```

If an Apache Tomcat server is placed behind a firewall, you must also define the `TomcatPortNumber` attribute. Set this attribute to the value of the `rmiServerPortPlatform` attribute as defined in the Tomcat JMX Lifecycle listener page of the Apache website.

Configuring user authentication for Worklight Server administration

You configure user authentication and choose an authentication method. Configuration procedure depends on the web application server that you use.

The Worklight Server administration requires user authentication.

You must perform some configuration after the installer deploys the Worklight Server administration web applications in the web application server.

The Worklight Server administration has the following Java Platform, Enterprise Edition (Java EE) security roles defined:
- `worklightadmin`
- `worklightdeployer`
- `worklightmonitor`
- `worklightoperator`

You must map the roles to the corresponding sets of users.

If you choose to use an authentication method through a user repository such as LDAP, you can configure the Worklight Server administration so that you can use users and groups with the user repository to define the Access Control List (ACL) of the Worklight Server administration. This procedure is conditioned by the type and version of the web application server that you use.

Configuring WebSphere Application Server full profile for Worklight Server administration:

Configure security by mapping the Worklight Server administration 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 `worklightadmin`, `worklightdeployer`, `worklightmonitor`, and `worklightoperator` 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**.
   d. Select **IBM_Worklight_Administration_Services**.
   e. In the **Configuration** tab, select **Details** > **Security role to user/group mapping**.

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_Worklight_Console**.

i. Click **Save** to save the changes.

Configuring WebSphere Application Server Liberty profile for Worklight Server administration:

Configure the Java EE security roles of the Worklight Server administration and the data source in the `server.xml` file.

Before you begin

In WebSphere Application Server Liberty profile, you configure the roles of **worklightadmin**, **worklightdeployer**, **worklightmonitor**, and **worklightoperator** 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 `<security-role>` elements. Each `<security-role>` element is for each roles: **worklightadmin**, **worklightdeployer**, **worklightmonitor**, and **worklightoperator**. Map the roles to the appropriate user group name, in this example: **worklightadmingroup**, **worklightdeployergroup**, **worklightmonitorgroup**, or **worklightoperatorgroup**. 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 administration database.

Procedure

1. Edit the `server.xml` file.

   For example:

   ```
   <security-role name="worklightadmin">
     <group name="worklightadmingroup"/>
   </security-role>
   <security-role name="worklightdeployer">
     <group name="worklightdeployergroup"/>
   </security-role>
   <security-role name="worklightmonitor">
     <group name="worklightmonitorgroup"/>
   </security-role>
   <security-role name="worklightoperator>
     <group name="worklightoperatorgroup"/>
   </security-role>

   <basicRegistry id="worklightadmin">
     <user name="admin" password="admin"/>
     <user name="guest" password="guest"/>
     <user name="demo" password="demo"/>
     <group name="worklightadmingroup">
       <member name="guest"/>
       <member name="demo"/>
     </group>
   ```

```
        <group name="worklightdeployergroup">
          <member name="admin" id="admin"/>
        </group>
        <group name="worklightmonitorgroup"/>
        <group name="worklightoperatorgroup"/>
      </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="WLADMIN" jndiName="jdbc/WorklightAdminDS" connectionManagerRef="AppCenterPool"
 ...
</dataSource>
```

Configuring Apache Tomcat for Worklight Server administration:

You must configure the JEE security roles for the Worklight Server administration
on the Apache Tomcat web application server.

Procedure

1. If you installed the Worklight Server administration manually, declare the
   following roles in the `conf/tomcat-users.xml` file.

```
<role rolename="worklightadmin"/>
<role rolename="worklightmonitor"/>
<role rolename="worklightdeployer"/>
<role rolename="worklightoperator"/>
```

2. Add roles to the selected users, for example:

```
<user name="demo" password="demo" roles="worklightadmin"/>
```

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

Defining the endpoint of the Worklight Administration services

If circumstances require the parameters of the endpoint definition to be changed,
you must configure properties of the web application server appropriately.

Worklight Console must be able to locate the Worklight Administration REST
services and must be able to generate various URI for the entry points of web
applications or for the download of audit log files.

By default, the URI protocol, hostname, and port are the same as those defined in
the web application server used to access Worklight Console; the context root of
the Worklight Administration REST services is `worklightadmin`. When the context
root of the Worklight Administration REST services is changed or when the
internal URI of the web application server is different from the external URI, and
the external URI is used to access Worklight Console, the externally accessible
endpoint (protocol, hostname, and port) 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...) when Worklight Console is accessed with the
external address (wrklght.net).
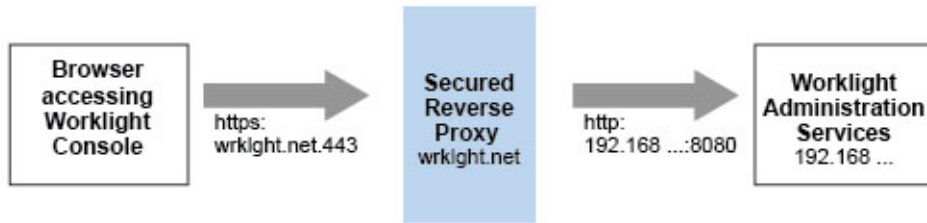
Figure 5. Configuration with secured reverse proxy

Table 36. The endpoint properties

Property name	Purpose	Example
`ibm.worklight.admin.endpoint`	This property enables Worklight Console to locate the Worklight Administration REST services. The value of this property must be specified as the external address and context root of the `worklightadmin.war` web application. You can use the asterisk (*) character as wildcard for specifying that the Worklight Administration services use the same value as Worklight Console. For example: `*://*:*/wladmin` means use the same protocol, host, and port as Worklight Console, but use `wladmin` as context root. This property must be specified for the Worklight Console application.	https://wrklght.net:443/ worklightadmin
`ibm.worklight.admin.proxy.protocol`	If external access is required, this property specifies the protocol for external browsers to access the Worklight Administration services. This property must be specified for the Worklight Administration services application.	https
`ibm.worklight.admin.proxy.host`	If external access is required, this property specifies the hostname for external browsers to access the Worklight Administration services. This property must be specified for the Worklight Administration services application.	wrklght.net

Table 36. The endpoint properties (continued)

Property name	Purpose	Example
`ibm.worklight.admin.proxy.port`	If external access is required, this property specifies the port for external browsers to access the Worklight Administration services. This property must be specified for the Worklight Administration services application.	443

Configuring the endpoint (WebSphere Application Server full profile):

Configure the endpoint of the application resources in the environment entries of Worklight Console and the Worklight Administration services application.

About this task

Follow this procedure when you must change the endpoint of the Worklight Administration services.

Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Applications** > **Application Types** > **WebSphere enterprise applications**.
3. Click **Worklight Administration 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.worklight.admin.proxy.host`, assign the hostname.
   b. For `ibm.worklight.admin.proxy.port`, assign the port number.
   c. For `ibm.worklight.admin.proxy.protocol`, assign the external protocol.
6. Click **OK** and save the configuration.
7. Select **Applications** > **Application Types** > **WebSphere enterprise applications**.
8. Click **Worklight Console**.
9. In the "Web Module Properties" section, select "Environment entries for Web modules".
10. For `ibm.worklight.admin.endpoint`, assign the full URI of the Worklight Administration services; That is, 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 for specifying that the Worklight Administration services use the same value as Worklight Console. For example, *://*:*/wladmin means use the same protocol, host, and port as Worklight Console, but use `wladmin` as context root.
11. Click **OK** and save the configuration. For a complete list of JNDI properties that you can set, see "List of JNDI properties for Worklight Server administration" on page 158.

Configuring the endpoint (Liberty profile):

For the Liberty profile, configure the endpoint of the application resources through the JNDI environment.

About this task

Follow this procedure when you must change the endpoint of Worklight Administration services. The appropriate entries in the `server.xml` file must be correctly defined.

Procedure

1. Ensure that the `<feature>` element in the `server.xml` file is correctly defined to be able to define JNDI entries.

   `<feature>jndi-1.0</feature>`

2. In the `<server>` section of the `server.xml` file, add an entry for each required property. Each such entry should have the following syntax:

   `<jndiEntry jndiName="`*JNDI_property_name*`" value="`*property_value*`"/>`

   Where:

   - *JNDI_property_name* is the name of the property that you are adding.
   - *property_value* is the value of the property that you are adding.

   For a complete list of JNDI properties that you can set, see "List of JNDI properties for Worklight Server administration" on page 158.

Example of setting properties for configuring the endpoint

This example shows the settings of the properties in the `server.xml` file that are required for configuring the endpoint of the application resources.

```
worklightconsole/ibm.worklight.admin.endpoint
    value="https://wrklght.net:443/worklightadmin" />
<jndiEntry jndiName="worklightadmin/ibm.worklight.admin.proxy.protocol"
    value="https" />
<jndiEntry jndiName="worklightadmin/ibm.worklight.admin.proxy.host"
    value="wrklght.net" />
<jndiEntry jndiName="worklightadmin/ibm.worklight.admin.proxy.port"
    value="443" />
```

In this example, assume that the context root of Worklight Console is `worklightconsole` and that the context root of the Administration Services is `worklightadmin`. You can prefix the JNDI properties with the context root of the corresponding web application. If multiple instances of Worklight Server are running in the same web application server, this technique is particularly useful. If you have only one instance of Worklight Server, you can omit the context root prefix; for example:

```
<jndiEntry jndiName="ibm.worklight.admin.endpoint"
    value="https://wrklght.net:443/worklightadmin" />
<jndiEntry jndiName="ibm.worklight.admin.proxy.protocol"
    value="https" />
<jndiEntry jndiName="ibm.worklight.admin.proxy.host"
    value="wrklght.net" />
<jndiEntry jndiName="ibm.worklight.admin.proxy.port"
value="443"/>
```

For **ibm.worklight.admin.endpoint**, you can use the asterisk (*) character as wildcard for specifying that the Worklight Administration services use the same value as Worklight Console. For example, `*://*:*/wladmin` means use the same

protocol, host, and port as Worklight Console, but use `wladmin` as context root.

Configuring the endpoint (Apache Tomcat):

For the Apache Tomcat server, configure the endpoint of the application resources in the `server.xml` file.

About this task

Follow this procedure when you must change the endpoint of the Worklight Administration services. You must edit the `server.xml` file in the `conf` directory of your Apache Tomcat installation.

Procedure

In 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. Each 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 that you are adding.
- *property_value* is the value of the property that you are adding.
- *property_type* is the value of the type of property that you are adding.

For a complete list of JNDI properties that you can set, see "List of JNDI properties for Worklight Server administration" on page 158.
For **ibm.worklight.admin.endpoint**, you can use the asterisk (*) character as wildcard for specifying that the Worklight Administration services use the same value as Worklight Console. For example, `*://*:*/wladmin` means use the same protocol, host, and port as Worklight Console, but use `wladmin` as context root.

Example of setting server.xml properties for configuring the endpoint

This example shows the settings of the properties in the `server.xml` file that are required for configuring the endpoint of the application resources.

In the context section of the Worklight Console application:

```
<Environment name="ibm.worklight.admin.endpoint" value="https://wrklght.net:443/worklightadmin"
    type="java.lang.String" override="false"/></p>
```

You can use the asterisk (*) character as wildcard for specifying that the Worklight Administration services use the same value as Worklight Console. For example, `*://*:*/wladmin` means use the same protocol, host, and port as Worklight Console, but use `wladmin` as context root.

In the `<context>` section of the Worklight Administration Services application, you can write:

```
<Environment name="ibm.worklight.admin.proxy.protocol" value="https" type="java.lang.String"
    override="false"/>
<Environment name="ibm.worklight.admin.proxy.host" value="wrklght.net" type="java.lang.String"
    override="false"/>
<Environment name="ibm.worklight.admin.proxy.port" value="443" type="java.lang.Integer" override="
```

List of JNDI properties for Worklight Server administration

When you configure Worklight Server Administration Services and Worklight Console for your application server, you set optional or mandatory JNDI properties, in particular for Java Management Extensions (JMX).

Table 37. JNDI properties for Administration Services: JMX

Property	Optional/ Mandatory	Description	Restrictions
`ibm.worklight.admin.jmx.connector`	Optional	The Java Management Extensions (JMX) connector type. The possible values are SOAP and RMI. The default value is SOAP.	WebSphere Application Server only.
`ibm.worklight.admin.jmx.host`	Optional	Host name for the JMX REST connection.	Liberty profile only.
`ibm.worklight.admin.jmx.port`	Optional	Port for the JMX REST connection.	Liberty profile only.
`ibm.worklight.admin.jmx.user`	Optional	User name for the JMX REST connection.	Liberty profile only.
`ibm.worklight.admin.jmx.pwd`	Optional	User password for the JMX REST connection.	Liberty profile only.
`ibm.worklight.admin.rmi.registryPort`	Optional	RMI registry port for the JMX connection through a firewall.	Tomcat only.
`ibm.worklight.admin.rmi.serverPort`	Optional	RMI server port for the JMX connection through a firewall.	Tomcat only.
`ibm.worklight.admin.jmx.dmgr.host`	Mandatory	Deployment manager host name.	WebSphere Application Server Network Deployment only.
`ibm.worklight.admin.jmx.dmgr.port`	Mandatory	Deployment manager RMI or SOAP port.	WebSphere Application Server Network Deployment only.

Table 38. JNDI properties for Administration Services: time out

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.actions.prepareTimeout`	Optional	Timeout in milliseconds to transfer data from the management service to the runtime during a deployment transaction. If the runtime cannot be reached within this time, an error is raised and the deployment transaction ends. Default value: 1800000 ms (30 min)

Table 38. JNDI properties for Administration Services: time out (continued)

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.actions.commitRejectTimeout`		Timeout in milliseconds, when a runtime is contacted, to commit or reject a deployment transaction. If the runtime cannot be reached within this time, an error is raised and the deployment transaction ends. Default value: 120000 ms (2 min)
`ibm.worklight.admin.lockTimeoutInMillis`	Optional	Timeout in milliseconds for obtaining the transaction lock. Because deployment transactions run sequentially, they use a lock. Therefore, a transaction must wait until a previous transaction is finished. This timeout is the maximal time during which a transaction waits. Default value: 1200000 ms (20 min)
`ibm.worklight.admin.maxLockTimeInMillis`	Optional	The maximal time during which a process can take the transaction lock. Because deployment transactions run sequentially, they use a lock. If the application server fails while a lock is taken, it can happen in rare situations that the lock is not released at the next restart of the application server. In this case, the lock is released automatically after the maximum lock time so that the server is not blocked forever. Set a time that is longer than a normal transaction. Default value: 1800000 (30 min)

Table 39. JNDI properties for Administration Services: logging

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.logging.formatjson`	Optional	Set this property to `true` to enable pretty formatting (extra blank space) of JSON objects in responses and log messages. Setting this property is helpful when you debug the server. Default value: `false`.
`ibm.worklight.admin.logging.tosystemerror`	Optional	Specifies whether all logging messages are also directed to `System.Error`. Setting this property is helpful when you debug the server.

Table 40. JNDI properties for Administration Services: proxies

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.proxy.port`	Optional	If the Worklight Administration server is behind a firewall or reverse proxy, this property specifies the address of the host. Set this property to enable a user outside the firewall to reach the Worklight Administration server. Typically, this property is the port of the proxy, for example 443. It is necessary only if the protocol of the external and internal URIs are different.
`ibm.worklight.admin.proxy.protocol`	Optional	If the Worklight Administration server is behind a firewall or reverse proxy, this property specifies the protocol (HTTP or HTTPS). Set this property to enable a user outside the firewall to reach the Worklight Administration server. Typically, this property is set to the protocol of the proxy. For example, `wl.net`. This property is necessary only if the protocol of the external and internal URIs are different.
`ibm.worklight.admin.proxy.scheme`	Optional	This property is just an alternative name for `ibm.worklight.admin.proxy.protocol`.
`ibm.worklight.admin.proxy.host`	Optional	If the Worklight Administration server is behind a firewall or reverse proxy, this property specifies the address of the host. Set this property to enable a user outside the firewall to reach the Worklight Administration server. Typically, this property is the address of the proxy.

Table 41. JNDI properties for Administration Services: topologies and connections

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.audit`	Optional.	Set this property to `false` to disable the audit feature of the Worklight Server Administration console. The default value is `true`.
`ibm.worklight.admin.environmentid`	Optional.	Environment identifier for the registration of the MBeans. Use this identifier when different instances of the Worklight Server are installed on the same application server. The identifier determines which Administration Services, which console, and which runtimes belong to the same installation. The Administration Services manage only the runtimes that have the same environment identifier.
`ibm.worklight.admin.serverid`	Optional.	Server identifier. Must be different for each server in the farm. For server farms only.

Table 41. JNDI properties for Administration Services: topologies and connections (continued)

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.hsts`	Optional.	Set to true to enable HTTP Strict Transport Security according to RFC 6797.
`ibm.worklight.admin.db.jndi.name`	Optional	The JNDI name of the database. This parameter is the normal mechanism to specify the database. The default value is `java:comp/env/jdbc/WorklightAdminDS`.
`ibm.worklight.admin.db.type`	Optional Conditionally mandatory	The database type. Mandatory when the database is not specified by the `ibm.worklight.admin.db.jndi.name` property.
`ibm.worklight.admin.db.openjpa.ConnectionDriverName`	Conditionally mandatory	The fully qualified name of the database connection driver class. Mandatory only when the data source that is specified by the `ibm.worklight.admin.db.jndi.name` property is not defined in the application server configuration.
`ibm.worklight.admin.db.openjpa.ConnectionURL`	Conditionally mandatory	The URL for the database connection. Mandatory only when the data source that is specified by the `ibm.worklight.admin.db.jndi.name` property is not defined in the application server configuration.
`ibm.worklight.admin.db.openjpa.ConnectionUserName`	Conditionally mandatory	The user name for the database connection. Mandatory only when the data source that is specified by the `ibm.worklight.admin.db.jndi.name` property is not defined in the application server configuration.
`ibm.worklight.admin.db.openjpa.ConnectionPassword`	Conditionally mandatory	The password for the database connection. Mandatory only when the data source that is specified by the `ibm.worklight.admin.db.jndi.name` property is not defined in the application server configuration.
`ibm.worklight.admin.db.openjpa.Log`	Optional	This property is passed to OpenJPA and enables JPA logging. For more information, see the Apache OpenJPA User's Guide.
`ibm.worklight.topology.platform`	Not strictly mandatory but your application works best if you set this property.	Server type. Valid values: • `Liberty` • `WAS` • `Tomcat` If you do not set the value, the application tries to guess the server type.

Table 41. JNDI properties for Administration Services: topologies and connections (continued)

Property	Optional/ Mandatory	Description
`ibm.worklight.topology.clustermode`	Not strictly mandatory but your application works best if you set this property.	In addition to the server type, specify here the server topology. Valid values: • `Standalone` • `Cluster` • `Farm` The value by default is `Standalone`.
`ibm.worklight.farm.type`	Optional.	This property is for server farms only and sets the type of the node farm if a node farm is to be used. Only the `FILE` type is currently supported.
`ibm.worklight.farm.definition.location`	Optional Conditionally mandatory	This property is for server farm only and sets the full path to the XML configuration file of the server farm. The configuration file is checked against the `FarmSchema.xsd` schema. This option is mandatory for the `FILE` type of server farm.

Table 42. JNDI properties for the Worklight Console

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.endpoint`	Optional	Enables the Worklight console to locate the Worklight Server Administration REST services. Specify the external address and context root of the `worklightadmin.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://wl.net:443/worklightadmin`.
`ibm.worklight.admin.global.logout`	Optional	Clears the WebSphere user authentication cache during the console logout. This property is useful only for WebSphere Application Server V7. The default value is `false`.
`ibm.worklight.admin.hsts`	Optional	Set this property to `true` to enable HTTP Strict Transport Security according to RFC 6797. For more information, see the W3C Strict Transport Security page. The default value is `false`.
`ibm.worklight.admin.ui.cors`	Optional	The default value is `true`. For more information, see the W3C Cross-Origin Resource Sharing page.

Table 42. JNDI properties for the Worklight Console (continued)

Property	Optional/ Mandatory	Description
`ibm.worklight.admin.ui.cors.strictssl`	Optional	Set to `false` to allow CORS situations where the Worklight Console is secured with SSL (HTTPS protocol) while the Worklight Server Administration services are not, or conversely. This property takes effect only if the `ibm.worklight.admin.ui.cors` property is enabled.

Configuring the JNDI properties

For more information about how to configure the JNDI properties, see the topic "Configuring a Worklight project in production by using JNDI environment entries" on page 1182, sections *Configuring with the Ant task*, *Manually configuring on the server*, and *Procedure*.

To configure the properties with Ant tasks, you must use the Ant task `installworklightadmin`, instead of `configureapplicationserver`.

For the console, the property element should be under the console element.

For more information, see "Ant tasks for installation of Worklight Console and Administration Services" on page 120.

Verifying the installation of Worklight Server administration

You must log in to the Worklight Console to verify that the installation was successful.

Procedure

1. Open a web browser.
2. Enter the following URL in the address bar: `http://hostname:9083/ worklightconsole/`.

   Note:
   - In this URL, *hostname* is the host name of the computer that runs your application server.
   - You must replace *9083* by the HTTP port of your application server.
3. Log in with a `worklightadmin` user role.
4. The console displays the following message: **No runtime can be found.**. To install a Worklight runtime environment that you can manage with the Worklight Console, see "Installing the Worklight runtime environment" on page 164.

   Note: If the Application Center and Worklight Console are installed in the same Tomcat instance, you cannot log in to the Application Center console and to the Worklight Console at the same time in the same browser. If you try to log in at the same time, you get a `404 Page Not Found` error message.

   For example, you get this error message if you open your browser, successfully log in to the Application Center console, open a new tab in the browser, and log in to the Worklight Console.

This is a technical limitation of Tomcat. The implementation of single sign-on in Tomcat does not allow to use the same browser to log in to the Application Center console and to the Worklight Console at the same time. But the Application Center and Worklight Server require single sign-on. You must exit the browser after your work is done in the Application Center console and restart the browser to log in to the Worklight Console. You can then successfully log in to the Worklight Console.

Installing the Worklight runtime environment

About this task

For more information about the Worklight runtime environment, see "Deploying the project WAR file" on page 1105.

Installing a server farm

Worklight Server provides a specific plug-in so that instances of application servers can become server farm nodes. After you have prepared the installation depending on your work environment, you can install your server farm manually or by running Ant tasks.

Planning the configuration of a server farm

To plan the configuration of a server farm, choose the application server, write the configuration file, and deploy the WAR files.

In IBM Worklight Foundation, a server farm is composed of multiple stand-alone application servers that are not federated, or administered by a managing component of an application server. Worklight Server internally provides a farm plug-in as the means to enhance an application server so that it can be part of a server farm.

When to declare a server farm

Declare a server farm in the following cases:
- IBM Worklight Foundation is installed on multiple Tomcat application servers.
- IBM Worklight Foundation is installed on multiple WebSphere Application Server servers but not on WebSphere Application Server Network Deployer.
- IBM Worklight Foundation is installed on multiple WebSphere Application Server Liberty servers.

Do not declare a server farm in the following cases:
- Your application server is stand-alone.
- Multiple application servers are federated by WebSphere Application Server Network Deployment.

Work flow

When you plan to set up a server farm, you first choose an application server and then write the appropriate configuration file.
1. Choose a stand-alone application server for the server farm. IBM Worklight Foundation supports the following applications servers for server farms:
   - WebSphere Application Server Base
   - WebSphere Application Server Liberty

- Apache Tomcat

Note: To know which versions of application servers are supported, see "System requirements for using IBM Worklight Foundation" on page 13.
Attention: IBM Worklight Foundation V6.2.0 supports only homogeneous server farms. A server farm is said to be homogeneous when it connects application servers of the same type. Attempting to associate different types of application servers would lead to unpredictable behavior at run time. For example, a farm of Tomcat servers and WebSphere Application Server servers is an invalid configuration.

2. After you chose an application server, write a configuration file for your server farm.

   This configuration file is a flat file in XML format, which specifies the connectivity settings for each server. The farm plug-in reads the file at run time so that each server can communicate with all others within the farm. Each server in the farm uses the same configuration file at run time. Therefore, this file can either be duplicated on the file system of each server or be placed in a shared directory, with appropriate read rights for each server.

   For more information about this configuration file, see "Defining a server farm for Worklight Server administration" on page 149.

3. When you plan to deploy Worklight runtime environments on a farm of N servers, deploy the following elements on each of the N servers, according to the Worklight symmetric deployment scheme.
   - The WAR file for Worklight Administration Services
   - The WAR file for each Worklight runtime environment

   The databases for Worklight Administration Services, Worklight runtime environments, and Worklight reports are shared among all the servers.

   Note: Therefore, you cannot use Derby databases in a server farm because they allow only a single connection at a time.

4. Exchange the server certificates in their truststores.

   This exchange is mandatory for WebSphere Application Server and Liberty farms because communication is secured with SSL between the server. The exchange is optional for Tomcat farms.

Installing a server farm by using an Ant task

To install a server farm, you can use an Ant task. You stop the servers, write a configuration file, customize the Ant script, and run it so that each server becomes a farm node. Then, you add the necessary JNDI properties and exchange the signer certificates between the servers.

Before you begin

Make sure to stop all the servers that you are about to configure as members of a server farm.

Configure the REST connector of all the Liberty servers of the farm, by following the procedure at "Configuring WebSphere Application Server Liberty profile" on page 107. This configuration defines the following parameters for each Liberty server:
- The Liberty administrator and password, which are required to define the configuration farm plug-in in step 1.
- The keystore, which is required to exchange public certificates in step 5.

Procedure

1. Write the configuration file for the farm plug-in.

   The configuration file is a flat file in XML format. You must write it manually. In this file, each server is defined as an XML element with attributes that are specific to each type of supported application server: WebSphere Application Server Express or Base, Liberty, or Tomcat. These attributes identify a server as unique, indicate the connection type and ports to use, and provide credentials for authentication purpose when applicable. Your configuration file is read by the farm plug-in at run time and validated against the `FarmSchema.xsd` XML schema. When you write the configuration file, you can use the schema to ensure that your XML code complies with it to prevent any problem at run time.

   Tip: Give your configuration files meaningful names, for example `TomcatFarm.xml` or `LibertyFarm.xml`.

   The contents of the configuration file depends on the type of application server. For instructions on how to write a farm plug-in configuration file:

   - For WebSphere Application Server, see step 2 on page 170 in "Installing a WebSphere Application Server farm manually" on page 170.
   - For Liberty, see step 2 on page 174 in "Installing a Liberty server farm manually" on page 173.
   - For Tomcat, see step 2 on page 178 in "Installing a Tomcat server farm manually" on page 177.

2. Customize your Ant script to configure the Worklight databases.

   a. If you must create the database, do as follows:

      - Edit the Ant script that you use later to configure the databases.

        To help you write configuration files, sample Ant files that use these Ant tasks are provided in `<WORKLIGHT_INSTALL_DIR>`/WorklightServer/ configuration-samples. The files are named after this pattern: `create-database-<database>.xml`. For more information, see Table 1 in "Sample configuration files" on page 1156.

      - To create the Administration Services databases, run the following command:

        `ant -f create-database-<database>.xml admdatabases`

      - If only one Worklight project is deployed in the farm, run the following command to create the Worklight project databases:

        `ant -f create-database-<database>.xml databases`

      - If several Worklight projects are to be deployed in the farm, each Worklight project must have its own database. Create a copy of the Ant file per project. Each one must point to a different database or schema, or, for Oracle, have a different user. Then, run the database target for each file.

      Because all the servers in the farm must share Worklight databases, you run these commands only once, even if you defined several servers.

   b. If the databases are created, and you must only create the database `TABLES`:

      - Edit the Ant script that you use later to create and configure the databases.

        To help you write configuration files, sample Ant files that use these Ant tasks are provided in `<WORKLIGHT_INSTALL_DIR>`/WorklightServer/ configuration-samples. The files are named after this pattern:

configure-*\<appServer>*-*\<database>*.xml. For more information, see Table 1 in "Sample configuration files" on page 1156.

- To create the Administration Services databases, run the following command:

  ant -f configure-*\<appServer>*-*\<database>*.xml admdatabases

- If only one Worklight project is deployed in the farm, run the following command to create the Worklight project databases:

  ant -f configure-*\<appServer>*-*\<database>*.xml databases

- If several Worklight projects are to be deployed in the farm, each Worklight project must have its own database. Create a copy of the Ant file per project. Each one must point to a different database or schema, or, for Oracle, have a different user. Then, run the database target for each file.

Because all the servers in the farm must share Worklight databases, you run these commands only once, even if you defined several servers.

3. Customize and run your Ant script so that each server becomes a farm node.

   For each server that you want to promote as a farm node, proceed as follows:

   a. Edit the Ant script that you use to configure the server.

      To install the Worklight Administration Services and the Worklight runtime environments, the script must contain at least the **admininstall** and **install** targets.

      To help you write configuration files, sample Ant files that use these Ant tasks are provided in *\<WORKLIGHT_INSTALL_DIR>*/WorklightServer/ configuration-samples. The files are named after this pattern: configure-*\<appServer>*-*\<database>*. For more information, see Table 1 in "Sample configuration files" on page 1156.

   b. For WebSphere Application Server Liberty profile, modify the JMX tag to define the Liberty administrator credentials. In the Ant file, replace the empty <jmx/> tag by the following:

      <jmx libertyAdminUser="demo" libertyAdminPassword="demo" createLibertyAdmin="false"/>

      Where:
      - **libertyAdminUser** is the name of the Liberty administrator.
      - **libertyAdminPassword** is the password of the Liberty administrator.
      - If **createLibertyAdmin** is set to false, the Ant task does not attempt to add the user in the basic registry, or to declare it as a Liberty administrator.

   c. Optional: In addition, if your server farm includes several Tomcat servers on the same computer, specify a different JMX port number for each server by setting the **tomcatRMIPort** attribute of the <jmx> element inside the **installworklightadmin** Ant task.

      This port number must match the value of the **JMXPortNumber** attribute of the corresponding <TomcatNode> element in the configuration file that you wrote in step 1 on page 166.

      For example, consider a farm with several Tomcat servers, two of which are on the same computer. Assign port 8686 to the first server and 8687 to the second.

      1) In the Ant script, add the following elements.
         - For the first server: <jmx tomcatRMIPort="8686"/>
         - For the second server:<jmx tomcatRMIPort="8687"/>
      2) In the configuration file of the farm, add the following elements.

```
<TomcatNode .. JMXPortNumber="8686">
<TomcatNode .. JMXPortNumber="8687">
```

    d. Deploy the Worklight Administration Services and Worklight runtime environments on the targeted application server.

       Run the following commands for each server of the farm.

```
ant -f configure-<appServer>-<database>.xml adminstall
ant -f configure-<appServer>-<database>.xml install
```

4. Manually add the necessary JNDI properties.

   Depending on your type of application server, proceed as follows:

- For a Tomcat farm or a WebSphere Application Server farm, set the JNDI properties for Worklight Administration Services for each server, as shown in Table 1.

Table 43. Tomcat or WebSphere Application Server farm: JNDI properties for Worklight Administration Services

Name	Value
`ibm.worklight.farm.type`	File
`ibm.worklight.farm.definition.location`	The absolute path to the XML configuration file for the farm plug-in. For example: `TomcatFile.xml` or `WasFile.xml`
`ibm.worklight.admin.serverid`	A unique identifier for each server. This identifier must match the corresponding **ServerID** attribute of the server element in the configuration file for the server that you are configuring (`<TomcatNode>` or `<WasNode>`).
`ibm.worklight.topology.clustermode` **Note:** The Ant task created this entry with the value `Standalone`. You must modify this value.	Farm

- For a Tomcat farm or a WebSphere Application Server farm, set the JNDI properties for Worklight runtime environments for each server, as shown in Table 2.

Table 44. Tomcat or WebSphere Application Server farm: JNDI properties for Worklight runtime environments

Name	Value
`ibm.worklight.admin.serverid`	A unique identifier for each server. This identifier must match the corresponding **ServerID** attribute of the server element in the configuration file for the server that you are configuring (`<TomcatNode>` or `<WasNode>`).
`ibm.worklight.topology.clustermode` **Note:** The Ant task created this entry with the value `Standalone`. You must modify this value.	Farm

- For a Liberty farm, set the JNDI properties for Worklight Administration Services and Worklight runtime environments for each server, as shown in Table 3.

Table 45. Liberty farm: JNDI properties for Worklight Administration Services and Worklight runtime environments

Header	Header
`ibm.worklight.farm.type`	File
`ibm.worklight.farm.definition.location`	The absolute path to the XML configuration file for the farm plug-in. For example: `LibertyFile.xml`
`ibm.worklight.admin.serverid`	A unique identifier for each server. This identifier must match the corresponding **ServerID** attribute of the server element in the configuration file for the server that you are configuring (`<LibertyNode>`).
`ibm.worklight.topology.clustermode` **Note:** The Ant task created this entry with the value `Standalone`. You must modify this value.	Farm

For more information, see the following documentation.

- For a farm of WebSphere Application Server stand-alone servers, see how to set JNDI properties in "Installing a WebSphere Application Server farm manually" on page 170.
  - For Worklight Administration Services, see step 4 on page 171.
  - For Worklight runtime environments, see step 5 on page 172.
- For a Tomcat farm, see how to set JNDI properties in "Installing a Tomcat server farm manually" on page 177:
  - For Worklight Administration Services, see step 5 on page 179.
  - For Worklight runtime environments, see step 7 on page 180.

5. Exchange the public certificates between all the servers of the farm.
   - For a farm of WebSphere Application Server stand-alone servers, see step 6 on page 173 in "Installing a WebSphere Application Server farm manually" on page 170.
   - For a Liberty farm, see step 8 on page 177 in "Installing a Liberty server farm manually" on page 173.
   - For a Tomcat farm: By default, the sample configuration Ant scripts that are provided with Worklight Server configure only the Tomcat HTTP connector, which does not use SSL. Therefore, you have nothing to do.
6. Replicate the LTPA keystores across farm members.

   Note: For a Liberty farm, see step 9 in "Installing a Liberty server farm manually" on page 173.

What to do next

Start the servers.

Set up an IBM HTTP Server for Liberty. For more information, see "Setting up an IBM HTTP Server in an IBM WebSphere Application Server Liberty profile farm" on page 316.

Installing a WebSphere Application Server farm manually

To install a WebSphere Application Server server farm step by step, you create the database, write the configuration file, configure the runtime database, and the Worklight Console and administration web applications, and exchange signer certificates between truststores.

Before you begin

Make sure that you have defined your WebSphere Application Server profile.

About this task

To configure a farm of WebSphere Application Server servers, follow these steps:

1. Create the Worklight Administration Services database.
2. Write the configuration file for the farm plug-in.
3. Configure the Worklight runtime database.
4. Install the Worklight Console and Administration Services applications.
5. Configure the Worklight runtime environments.
6. Exchange signer certificates between server truststore for each server of the farm..

Tip: When you work in the wizard of the WebSphere Application Server administration console, where no step number is mentioned in the following procedure, click **Next** or click the subsequent step number wizard list.

Procedure

1. Create the Worklight Administration Services database.

   Note: Worklight databases are shared among the application servers of a farm, which has two consequences:
   - You create this database only once, whatever the number of servers.
   - You cannot use the Derby database in such a topology because this database allows only a single connection at a time.

   For more information for each database, see the following documentation.
   - "Setting up your DB2 database manually for the Worklight Server administration" on page 127
   - "Setting up your Oracle database manually for the Worklight Server administration" on page 138
   - "Setting up your MySQL database manually for the Worklight Server administration" on page 135

2. Write the configuration file for the farm plug-in.
   a. Create an XML file.

      Give it a meaningful name, for example `WasFarm.xml`.

      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="FarmSchema.xsd">
          <WasNode ServerID="server id" AdminPass="user password" AdminUser="user name"
                  Hostname="host name address" JMXPortNumber="port number"/>
      .....
      </Farm>
      ```

      Where:

- *server id* is the unique identifier of the server
- *user password* is the WebSphere Application Server password.
- user name is the WebSphere Application Server user name. This name must be in the WebSphere Application Server user registry but does not need to have a WebSphere Application Server role because the Worklight MBeans are not secured
- host name address is the host name of the stand-alone WebSphere Application Server server.
- port number is the SOAP port number of the stand-alone WebSphere Application Server server.

   Tip: The port number must be different for each server that is hosted on the same computer.

   There must be one <WasNode> element per server in the farm. For example:

   ```
   <?xml version="1.0" encoding="UTF-8"?>
   <Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
     <WasNode ServerID="ServerOne" AdminPass="admin" AdminUser="admin" Hostname="MyHostName"
     <WasNode ServerID="ServerTwo" AdminPass="admin" AdminUser="admin" Hostname="MyHostName"
   </Farm>
   ```

   The complete syntax of the elements and attributes of this XML file and the underlying XML schema is documented in "Defining a server farm for Worklight Server administration" on page 149.

   b. Copy the configuration file to each computer where WebSphere Application Server is installed or on a shared file system that all the servers of the farm can access.

3. Configure the Worklight runtime database.

   Important: In the context of a server farm, you must set the scope of the JDBC connection to server level.
   - "Configuring WebSphere Application Server for DB2 manually" on page 1124
   - "Configuring WebSphere Application Server for Oracle manually" on page 1138
   - "Configuring WebSphere Application Server for MySQL manually" on page 1133

   a. Follow the instructions for configuring your database.
   b. Log on to the WebSphere Application Server administration console.
   c. On the navigation pane, select **Security** > **Global security**.
   d. In **Authentication**, select **Java Authentication and Authorization Service** > **J2C authentication data**.
   e. Clear **Prefix new alias names with the node name of the cell (for compatibility with earlier releases)**.
   f. Click **Apply** and then **Save**.

4. Install the Worklight Console and Administration Services applications.

   a. Follow the instructions in "Configuring WebSphere Application Server for Worklight Server administration manually" on page 144.
   b. On the navigation pane, select **Applications** > **Application Types** > **Websphere enterprise applications**.
   c. Click **Install**.
   d. Browse to the *<Worklight_Install_Dir>*/WorklightServer/ worklightadmin.war file for Worklight Administration Services.
   e. Click **Next**.

    f. Select **Detailed** to show all installation options and parameters.

    g. Click **Next**, and then **Continue**.

    h. In Step 7 **Map resource references to resource**, enter `/jdbc/WorklightAdminDS`.

    i. In Step 9 **Map context roots for web modules**, enter `/worklightadmin`.

    j. In Step 10 **Map environment entries for Web Modules**, proceed as follows:

        1) Enter `WAS` next to **ibm.worklight.topology.platform**.

        2) Enter `Farm` next to **ibm.worklight.topology.clustermode**

        3) Enter `File` next to **ibm.worklight.farm.type**.

        4) Enter the server identifier next to **ibm.worklight.admin.serverid**.

            This identifier must be the same as the one that you declared in the configuration file in step 2 on page 170.

        5) Enter the location of the XML configuration file in **ibm.worklight.farm.definition.location**.

    k. In Step 11 **Map security roles to users or groups**, select the `worklightadmin` role and then select **Map Special Subjects** > **All Authenticated in Application's Realm**.

    l. After Step 15, click **Finish** and save the configuration.

    m. Click the application name.

    n. In Detail properties, click **Class loading and update detection**.

    o. Select **Class loaded with local class loader first (parent last)**.

    p. Click **OK**.

    q. Click the application name again.

    r. In **Module**, click **Manage Modules** and then click the module link.

    s. Select **Class loaded with local class loader first (parent last)**.

    t. Click **OK**, and then again **OK**.

    u. Save the configuration.

5. Configure the Worklight runtime environments.

    For a list of JNDI properties, see "List of JNDI properties for Worklight Server administration" on page 158.

    a. Log on to the WebSphere Application Server administration console and go through the next steps for each deployed Worklight application.

    b. Configure WebSphere Application Server manually. You must configure variables, custom properties, and class loader policies. See Configuring WebSphere Application Server manually.

    c. On the navigation pane, select **Applications** > **Application Types** > **Websphere enterprise applications**.

    d. Click the application name.

    e. In **Web Module Properties**, click **Environment entries for Web Modules** to display the JNDI properties.

    f. Enter **WAS** next to **ibm.worklight.topology.platform**.

    g. Enter `Farm` next to **ibm.worklight.topology.clustermode**.

    h. Enter **SOAP** or **RMI** next to **ibm.worklight.admin.jmx.connector**.

    i. Enter the server identifier for this server next to **ibm.worklight.admin.serverid**.

        This identifier must be the same as the one that you declared in the configuration file in the **ServerId** attribute of the `<WasNode>` element for this instance of WebSphere Application Server.

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

    k. Click the application name.

    l. Click **Startup behavior**.

    m. Set 100 in **Startup order**, or a higher value, to ensure that the Worklight runtime environment starts after the Worklight Administration Services.

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

6. Exchange signer certificates between server truststore for each server of the farm.

    a. Log on to the WebSphere Application Server administration console.

    b. On the navigation pane, select **Security** > **SSL certificate and key management**.

    c. In **Related Items**, click **Key stores and certificates**.

    d. From the **Keystore usages** drop-down list, make sure that **SSL keystores** is selected.

    e. Import the certificates from all the other WebSphere Application Server servers of the farm.

        1) Click the **NodeDefaultTrustStore**.

        2) In the Additional Properties section, click **Signer certificates**.

        3) Click **Retrieve from port**.

        4) For each other server of the farm

            a) Enter its host name or IP address in the **Host** field.

            b) Enter its HTTPS transport (SSL) port in the **Port** field.

            c) From the **SSL configuration for outbound connection** drop-down list, select **NodeDefaultSSLSettings**.

            d) Enter an alias for this signer certificate in the **Alias** field.

            e) Click **Retrieve signer information**.

            f) Review the information that is retrieved from the remote certificate then click **OK**.

    f. Click **Save**.

    g. Restart the server.

Installing a Liberty server farm manually

To install a Liberty server farm step by step, you create the database, write the configuration file, configure the runtime database, and the Worklight Console and administration web applications, set JNDI properties, and exchange signer certificates between truststores.

Before you begin

Configure the REST connector of all the Liberty servers of the farm, by following the procedure at "Configuring WebSphere Application Server Liberty profile" on page 107. This configuration defines the following parameters for each Liberty server:

- The Liberty administrator and password, which are required to define the configuration farm plug-in in step 2 on page 174, and the `admin.jmx.user/password` JNDI properties in step 6 on page 176.

- The keystore, which is required to exchange public certificates in step 8 on page 177.

About this task

To configure a farm of Liberty servers, follow these steps:

1. Create the Worklight Administration Services database.
2. Write the configuration file for the farm plug-in.
3. Configure the Worklight runtime database.
4. Configure the Worklight Console application.
5. Configure the Worklight Administration Services application.
6. Configure the JNDI properties.
7. Configure the Worklight runtime environments.
8. Exchange signer certificates between server truststores.
9. Replicate the LTPA keystores across farm members.

Procedure

1. Create the Worklight Administration Services database.

   Note: Worklight databases are shared among the application servers of a farm, which has two consequences:
   - You create this database only once, whatever the number of servers.
   - You cannot use the Derby database in such a topology because this database allows only a single connection at a time.

   For more information for each database, see the following documentation.
   - "Setting up your DB2 database manually for the Worklight Server administration" on page 127
   - "Setting up your Oracle database manually for the Worklight Server administration" on page 138
   - "Setting up your MySQL database manually for the Worklight Server administration" on page 135

2. Write the configuration file for the farm plug-in.

   a. Write an XML file.

      Give it a meaningful name, for example `LibertyFarm.xml`.

      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Fa
        <LibertyNode ServerID="server id" AdminPass="user password" AdminUser="user name" Hostname="
        .....
      </Farm>
      ```

      Where:
      - `server id` is the unique identifier of the server
      - `user password` is the password of the Liberty administrator that you must create to enable the Liberty REST Connector. For more information, see "Configuring WebSphere Application Server Liberty profile" on page 107.
      - `user name` is the user name of the Liberty administrator that you must create to enable the Liberty REST Connector. For more information, see "Configuring WebSphere Application Server Liberty profile" on page 107.
      - `host name address` is the host name of the Liberty server
      - `port number` is the HTTPS port number of the Liberty server

        Tip: The port number must be different for each server that is hosted on the same computer.

Create one <LibertyNode> element per server in the farm. For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="
   <LibertyNode ServerID="S1" AdminPass="demo" AdminUser="demo" Hostname="MyHostName" JMXPor
   <LibertyNode ServerID="S2" AdminPass="demo" AdminUser="demo" Hostname="MyHostName" JMXPor
</Farm>
```

The complete syntax of the elements and attributes of this XML file and the underlying XML schema is documented in "Defining a server farm for Worklight Server administration" on page 149.

   b. Copy the configuration file to each computer where Liberty is installed or on a shared file system that all the servers of the farm can access.

3. Configure the Worklight runtime database.

   You create this database only once, whatever the number of servers. For more information for each database, see the following documentation.

   • "Setting up your DB2 databases manually" on page 1121
   • "Setting up your Oracle databases manually" on page 1136
   • "Setting up your MySQL databases manually" on page 1131

4. Configure the Worklight Console application.

   a. Make a backup of the <Liberty_Install_Dir>/servers/<server_name>/ server.xml file.

   b. Adding the following lines in the server.xml file.

```
<application id="worklightconsole" name=" worklightconsole" location="<war file location>"
    <application-bnd>
        <security-role name="worklightadmin">
            <user name="demo"/>
        </security-role>
    </application-bnd>
</application>
```

   The <war file location> placeholder represents the relative or absolute path to the Worklight Console WAR file <Worklight_Install_Dir>/ WorklightServer/worklightconsole.war.

5. Configure the Worklight Administration Services application.

   a. Make a backup of the <Liberty_Install_Dir>/servers/<server_name>/ server.xml file.

   b. Adding the following lines in the server.xml file.

```
<jndiEntry jndiName="worklightconsole/ibm.worklight.admin.endpoint" value='"*://*:*/worklig
<application id="worklightadmin" name="worklightadmin" location="<war file location>" type=
        <application-bnd>
            <security-role name="worklightadmin">
                <user name="demo"/>
            </security-role>
        </application-bnd>
        <classloader delegation="parentLast">
            <privateLibrary>
                <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordu
            </privateLibrary>
        </classloader>
</application>
```

   The <war file location> placeholder represents the relative or absolute path to the Worklight Administrative Services WAR file <Worklight_Install_Dir>/WorklightServer/worklightadmin.war.

   c. Configure the data sources as described in the following documentation.

      • DB2: "Configuring Liberty profile for DB2 manually for Worklight Server administration" on page 128

- Oracle: "Configuring Liberty profile for Oracle manually for Worklight Server administration" on page 139
- MySQL: "Configuring Liberty profile for MySQL manually for Worklight Server administration" on page 135

6. Configure the JNDI properties

   For a list of JNDI properties, see "List of JNDI properties for Worklight Server administration" on page 158.

   a. Make a backup of the *<Liberty_Install_Dir>*/servers/*<server_name>*/ server.xml file.

   b. Adding the following lines in the server.xml file.
   ```
   <jndiEntry jndiName="ibm.worklight.topology.platform" value="Liberty"/>
   <jndiEntry jndiName="ibm.worklight.topology.clustermode" value="Farm"/>
   <jndiEntry jndiName="ibm.worklight.farm.type" value="File"/>
   <jndiEntry jndiName="ibm.worklight.farm.definition.location" value="<plugin xml file location>
   <jndiEntry jndiName="ibm.worklight.admin.serverid" value="<server id>"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.host" value="<hostname>"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.port" value="<server HTTPS port number>"/>
   <jndiEntry jndiName="ibm.worklight.admin.jmx.user" value="demo" />
   <jndiEntry jndiName="ibm.worklight.admin.jmx.pwd" value="demo"/>
   ```

   Where
   - *<plugin xml file location>* is the location of the configuration file that you wrote in step 2 on page 174.
   - *<hostname>* is the host name of this server. This value must match the **Hostname** attribute of the <LibertyNode> element that corresponds to this server in the configuration file.
   - *<server id>* must have the same value as the **ServerId** attribute for this server in the <LibertyNode> element of the configuration file.
   - *<server HTTPS port number>* is the value of the **httpsPort** attribute in the <httpEndpoint> entry for this server.

   Note:
   - The **ibm.worklight.admin.jmx.user** JNDI property must have the same value as the **AdminUser** attribute in the <LibertyNode> element of the configuration file.
   - The **ibm.worklight.admin.jmx.pwd** JNDI property must have the same value as the **AdminPass** attribute in the <LibertyNode> element of the configuration file.

   c. Configure the data sources as described in the following documentation.
   - DB2: "Configuring Liberty profile for DB2 manually for Worklight Server administration" on page 128
   - Oracle: "Configuring Liberty profile for Oracle manually for Worklight Server administration" on page 139
   - MySQL: "Configuring Liberty profile for MySQL manually for Worklight Server administration" on page 135

7. Configure the Worklight runtime environments.

   a. Follow the instructions in "Configuring the WebSphere Liberty Profile manually" on page 1164.

   b. Make a backup copy of the *Liberty_install_dir*/servers/*server_name*/ server.xml file.

   c. Edit the *Liberty_install_dir*/servers/*server_name*/server.xml file.

   d. Add the following lines in the server.xml file.

```
<jndiEntry jndiName="<runtime name>/publicWorkLightProtocol" value='"http"'/>
<jndiEntry jndiName="<runtime name>/publicWorkLightPort" value='"<http port>"'/>
```

The *<http port>* placeholder represents the port value of the **httpPort** attribute in the `<httpEndpoint>` element.

8. Exchange signer certificates between server truststores.

   You can configure the truststore by using such IBM utilities as KeyTool or iKeyman.

   a. Import the public certificates of the other servers of the farm into this truststore.

   b. Restart each instance of WebSphere Application Server so that this security configuration takes effect.

   For more information about KeyTool, see the KeyTool page of the IBM SDK documentation

   For more information about iKeyman, see the iKeyman page of the IBM SDK documentation.

9. Replicate the LTPA keystores across farm members.

   Note: This step is required for Single Sign On (SSO) to work.

   a. Start one of the farm member.

      In case of a default LTPA configuration, the Liberty server generates a LTPA keystore *Liberty_install_dir*/servers/server_name/resources/security/ltpa.keys after it has successfully started.

      For more information about how to customize your LTPA configuration, see Configuring LTPA on the Liberty profile.

   b. Copy this `ltpa.keys` file in the directory *Liberty_install_dir*/servers/server_name/resources/security of each farm member.

What to do next

Set up an IBM HTTP Server for Liberty. For more information, see "Setting up an IBM HTTP Server in an IBM WebSphere Application Server Liberty profile farm" on page 316.

Installing a Tomcat server farm manually

To install a Tomcat server farm step by step, you create the database, write the configuration file, configure JMX, configure the runtime database, and the Worklight Console and administration web applications, and declare users and roles.

About this task

To configure a farm of Tomcat servers, follow these steps:

1. Create the Worklight Administration Services database.
2. Write the configuration file for the farm plug-in.
3. Configure Java Management Extensions (JMX).
4. Configure the Worklight runtime database.
5. Install the Worklight Administration Services application.
6. Install the Worklight Console application.
7. Configure the Worklight runtime environments.
8. Declare the users and roles to manage the applications in the Worklight Console.

Procedure

1. Create the Worklight Administration Services database.

   Note: Worklight databases are shared among the application servers of a farm, which has two consequences:

   - You create this database only once, whatever the number of servers.
   - You cannot use the Derby database in such a topology because this database allows only a single connection at a time.

   For more information for each database, see the following documentation.

   - "Setting up your DB2 database manually for the Worklight Server administration" on page 127
   - "Setting up your Oracle database manually for the Worklight Server administration" on page 138
   - "Setting up your MySQL database manually for the Worklight Server administration" on page 135

2. Write the configuration file for the farm plug-in.

   a. Create an XML file.

      Give it a meaningful name, for example `TomcatFarm.xml`.

      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Fa
        <TomcatNode ServerID="server id" Hostname="host name address" JMXPortNumber="port number"/>
        .....
      </Farm>
      ```

      Where:

      - `server id` is the unique identifier of the server
      - `host name address` is the host name of the Tomcat server
      - `port number` is the RMI port number of the Tomcat server

         Tip: The port number must be different for each server that is hosted on the same computer.

      There must be one `<TomcatNode>` element per server in the farm. For example:

      ```
      <?xml version="1.0" encoding="UTF-8"?>
      <Farm xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="Fa
        <TomcatNode ServerID="ServerOne" Hostname="MyHostName" JMXPortNumber="8686"/>
        <TomcatNode ServerID="ServerTwo" Hostname="MyHostName" JMXPortNumber="8687"/>
      </Farm>
      ```

      The complete syntax of the elements and attributes of this XML file and the underlying XML schema is documented in "Defining a server farm for Worklight Server administration" on page 149.

   b. Copy the configuration file to each computer where Tomcat is installed or on a shared file system that all the servers of the farm can access.

3. Configure Java Management Extensions (JMX).

   You configure JMX for each Tomcat server.

   a. Make sure that the **CATALINA_OPTS** environment variable sets all these Java properties: First check that the number tha tyou set for*<RMI_port number>* is not already used on your system.

      ```
      -Djava.rmi.server.hostname=localhost
      -Dcom.sun.management.jmxremote.port=<RMI_port number>
      -Dcom.sun.management.jmxremote.authenticate=false
      -Dcom.sun.management.jmxremote.ssl=false
      ```

b. Optional: If you want to be able to inspect the MBeans through the jconsole tool of your Java SDK, add the **-Dcom.sun.management.jmxremote** property.

Add a setenv file in the Tomcat bin/ directory.

- For UNIX environments: setenv.sh

```
# ========================== setenv.sh ==========================
# Allow to inspect the MBeans through jconsole
CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote"
# Configure JMX.
CATALINA_OPTS="$CATALINA_OPTS -Djava.rmi.server.hostname=localhost"
CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.port=<RMI_port number>"
CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.authenticate=false"
CATALINA_OPTS="$CATALINA_OPTS -Dcom.sun.management.jmxremote.ssl=false"
```

- For Windows environments: setenv.bat

```
REM ========================== setenv.bat ==========================
REM Allow to inspect the MBeans through jconsole
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote
REM Configure JMX.
set CATALINA_OPTS=%CATALINA_OPTS% -Djava.rmi.server.hostname=localhost
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.port=<RMI_port number>
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.authenticate=false
set CATALINA_OPTS=%CATALINA_OPTS% -Dcom.sun.management.jmxremote.ssl=false
```

4. Configure the Worklight runtime database.

You create this database only once, whatever the number of servers. For more information for each database, see the following documentation.

- "Setting up your DB2 databases manually" on page 1121
- "Setting up your Oracle databases manually" on page 1136
- "Setting up your MySQL databases manually" on page 1131

5. Configure the Worklight Administration Services application.

a. Remove the *<Tomcat_install_dir>*/webapps/worklightadmin directory.

b. Copy the Worklight Administration Services file worklightadmin.war from *<Workligh_install_dir>*/WorklightServer to *<Tomcat_install_dir>*/webapps/.

c. Make a backup copy of the *<Tomcat_install_dir>*/conf/server.xml file.

d. Edit the *<Tomcat_install_dir>*/conf/server.xml file.

e. Add the following lines in the <Host> element:

```
<Context docBase="worklightadmin" path="/worklightadmin">
  <Resource ... />
  <Environment name="ibm.worklight.topology.platform" value="Tomcat" type="java.lang.String
  <Environment name="ibm.worklight.topology.clustermode" value="Farm" type="java.lang.Strin
  <Environment name="ibm.worklight.admin.serverid" value="<server ID>"
       type="java.lang.String" override="false"/>
  <Environment name="ibm.worklight.farm.type" value="File" type="java.lang.String" override
  <Environment name="ibm.worklight.farm.definition.location" value="<farm plugin xml file l
       type="java.lang.String" override="false"/>
</Context>
```

Where

- The attributes of the <Resource> element depend on the database. You can use the following examples in this documentation:
  - For DB2 databases, see "Configuring Apache Tomcat for DB2 manually for Worklight Server administration" on page 130.
  - For Oracle databases, see "Configuring Apache Tomcat for Oracle manually for Worklight Server administration" on page 142.
  - For MySQL databases, see "Configuring Apache Tomcat for MySQL manually for Worklight Server administration" on page 138.

Chapter 7. Installing and configuring **179**

- *<farm plugin xml file location>* is the absolute path to the configuration file that you wrote in step Write the configuration file for the farm plug-in..
- *<server ID>* is a unique identifier for this server. This identifier must match the **ServerID** attribute of the <TomcatNode> element that you declared in the configuration file for this server.

  Attention: The name of the web application WAR file (`worklightadmin.war`) must match the context root (/worklightadmin), otherwise ill effects might occur.

6. Configure the Worklight Console application.
   a. Remove the *<Tomcat_install_dir>*/webapps/worklightconsole directory.
   b. Copy the `worklightconsole.war` file to *<Tomcat_install_dir>*/webapps/.
   c. Make a backup copy of the *<Tomcat_install_dir>*/conf/server.xml file.
   d. Edit the *<Tomcat_install_dir>*/conf/server.xml file.
   e. Add the following lines into the <Host> element:

   ```
   <Context docBase="worklightconsole" path="/worklightconsole">
      <Environment name="ibm.worklight.admin.endpoint" value="*://*:*/worklightadmin"
                 type="java.lang.String" override="false"/>
   </Context>
   ```

   Attention: The name of the web application WAR file (`worklightadmin.war`) must match the context root (/worklightadmin), otherwise ill effects might occur.

7. Configure the Worklight runtime environments.
   a. Make a backup copy of the *<Tomcat_install_dir>*/conf/server.xml file.
   b. Edit the *<Tomcat_install_dir>*/conf/server.xml file.
   c. Add the following lines in the <Host> element:

   ```
   <Context docBase="<runtime>" path="/<runtime>">
      <Loader className="org.apache.catalina.loader.VirtualWebappLoader" virtualClasspath="${cata
      <Environment name="ibm.worklight.topology.platform" value="Tomcat" type="java.lang.String"
      <Environment name="ibm.worklight.topology.clustermode" value="farm" type="java.lang.String"
      <Environment name="ibm.worklight.admin.serverid" value="<server id>" type="java.lang.String"
      <Resource ... name="jdbc/WorklightDS" ... />
      <Resource ... name="jdbc/WorklightReportsDS" ... />
   ```

   Where
   - The attributes of the <Resource> element depend on the database. You can use the examples that are provided in the following documentation.
     - For DB2 databases, see "Configuring Apache Tomcat for DB2 manually" on page 1126.
     - For Oracle databases, see "Configuring Apache Tomcat for Oracle manually" on page 1140.
     - For MySQL databases, see "Configuring Apache Tomcat for MySQL manually" on page 1135.
   - <runtime> is the name of your runtime environment, that is, the name of the Worklight WAR file without its extension.
   - <server ID> is a unique identifier for this server. This identifier must match the **ServerID** attribute of the <TomcatNode> element as declared in the configuration file for this server.

8. Declare the users and roles to manage the applications in the Worklight Console .

a. Make a backup copy of the *<Tomcat_install_dir>*/conf/tomcat-users.xml file.

b. Edit the *<Tomcat_install_dir>*/conf/tomcat-users.xml file.

c. Add the following roles and users in the <tomcat-users> element.

```
<!-- Define roles and users for the IBM Worklight Admin webapps. -->
  <role name="worklightadmin"/>
  <role name="worklightdeployer"/>
  <role name="worklightmonitor"/>
  <role name="worklightoperator"/>
  <user name="admin" password="admin" roles="worklightadmin"/>
  <user name="demo" password="demo" roles="worklightadmin"/>
  <user name="deployer" password="deployer" roles="worklightdeployer"/>
  <user name="monitor" password="monitor" roles="worklightmonitor"/>
  <user name="operator" password="operator" roles="worklightoperator"/>
```

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 Worklight instance, for example in a source control system.

It is advisable to back up the 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 Java virtual machine (JVM) memory, HTTP connections, back-end connections, and internal settings.

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

JDK

The Worklight Server can run on IBM JDK or Oracle JDK.

JVM memory allocation

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

- Set the JVM memory to 2 GB at least.
- 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 the JVM memory options in the jvm.options file in the usr/servers/*server-name*/ directory. You must create this file if it does not exist.
  - WebSphere Application Server: proceed as follows.
    1. Log in to the administration console.
    2. Go to **Servers** > **Server types** > **WebSphere application servers**.

3. Select 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.
- In a WebSphere Application Server Network Deployment environment, the minimum heap size of the Deployment Manager JVM must be 512. To set a heap size for the Deployment Manager, follow these steps:
  1. In the navigation pane, click **System Administration** > **Deployment Manager**.
  2. Under **Server Infrastructure** > **Java and Process Management**, click **Process definition**.
  3. Under **Additional Properties**, click **Java Virtual Machine**.
  4. Set a value in the **Maximum Heap Size** field.
  5. Save the changes to the master repository.
  6. Restart all servers, node agents, and the deployment manager.

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 uses a thread. 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 Liberty profile: Configuration elements in the server.xml file in the WebSphere Application Server documentation.
- WebSphere Application Server: Proceed as follows:
  1. Log in to the administration console.
  2. Go to **Servers** > **Server types** > **WebSphere application servers** > *server_name* > **Web container**.
     By default, the maximum number of threads is 50.
- Apache Tomcat: See the The HTTP Connector page on the Apache Tomcat website. 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. For more information, 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. For finer tuning, 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. *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 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 timeout period that is specified for the server session.
- 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 deployable elements 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 any accounts are inactive 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 the value of the **serverSessionTimeout** property.

Optimization of Worklight Server project databases

You can improve the performance of the project databases or schemas that support Worklight Server.

The following sections provide general information about database tuning, and techniques you can use to optimize your database performance for IBM Worklight Foundation. 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 compute your hardware needs, consider servers that offer multiple disks because performance increases significantly if you use disks correctly when you set up your Worklight Server project databases. 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 database logs, index, and data. Multidisk configuration results in 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

By using the compression feature set by your database vendor, you can decrease database size and input/output (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, LOB data size

If your database is DB2, consider using the `INLINE_LENGTH` option when you create tables for SSO information. This option is also appropriate for tables that contain data that is stored as large objects (LOBs), but that are only a few kilobytes in size. To improve performance of LOB data access, you can constrain the LOB size by placing the LOB data within the formatted rows on data pages rather than 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. You can improve performance and the purging accumulated data by mapping each

table partition to a different table space. This suggestion applies only to the `APP_ACTIVITY_REPORT` table, which holds most of the row data.

Note: Partitioned tables are different from a partitioned database (DPF) environment, which is not suggested for use with IBM Worklight Foundation.

To show how to use database partitions can be used, here is an example from DB2:

- A partition is defined on the `ACTIVITY_TIMESTAMP` column in the `APP_ACTIVITY_REPORT` table.
- Each partition contains the data for one day.
- 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

After high-volume data is allocated to separate table spaces, the task of periodically purging the data is simplified. This suggestion is also primarily relevant only to the `APP_ACTIVITY_REPORT` table that holds most of the row data. The process in this DB2 example is as follows:

- Aggregate data either with a 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, you purge the data purge by detaching the partition to a temp table, then truncating that temp table and attaching a new day to the partition. You can implement the process as a scheduled stored procedure 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;
```

Testing Worklight Server performance

You can run performance tests on the different features of the Worklight Server. This section describes how to run the Apache jMeter performance test tool, but the procedure is similar for other tools.

The following features can have an impact on Worklight Server performance:
- Authentication flow
- Back-end invocation
- Database reporting
- Single sign-on (SSO)
- Direct update
- Push notification
- Geolocation

This section focuses on testing the impact of authentication flow and back-end invocation on Worklight Server performance.

Testing authentication flow performance

The following realms, which are part of the default security test for both Android and IOS, are tested:

Remote disable realm
> Check on every request that the application is not blocked.

AntiXSRF realm
> Check on every request that `WL-Instance-Id` is equal to the one sent in the `init` response.

Anonymous User realm
> Generate a random user ID that is used for such things as reports and identifying the user.

Device no provisioning
> Check that the token value inside the authorization header is equal to the one sent in the initialization response.

For more information about the realms, see "The authentication configuration file" on page 877.

When you run a performance test, your first step is to complete the authentication flow. If you do not do so, security challenges are raised and your requests are rejected with "401" errors. This step involves sending an `init` request to the Worklight Server and extracting the relevant data from the response. The `init`

request has the following structure: `http://{Host}:{Port}/{Context}/apps/ services/api/{AppName}/{environment}/init`

Table 46. Initialization parameters

Parameter	Description
`x-wl-app-version`	Application version.
`x-wl-platform-version`	Version of Worklight Studio that built the application.

This is an example of a jMeter test:

The dynamic parameters in the Form Data (`skinLoaderChecksum`, `isAjaxRequest` and x) are appended to the URL. During performance testing, the `skin` and `skinLoaderChecksum` parameters are not needed because jMeter does not really run the app: jMeter only simulates the client app. The parameter x aims to prevent response data from being returned from cache. As a result, you do not need to append the parameters during performance testing or you can generate a random value for the dynamic parameter x. A better option is always to clean cookies in your performance testing tool before you start loading test threads.

Response data from Worklight initialization service

The response data from the Worklight `init` request differs depending on the security test you apply on your Worklight application environment. By default, if you have no additional security test, the response data structure for the common and iPhone environment are shown in the following figures. (The data structure for the Android environment is the same as that for the iPhone environment.)

Figure 6. Response data from common environment

Figure 7. Response data from iPhone environment

The difference between the common and iphone environment data structures is that the common environment has no `wl_deviceNoProvisioningRealm` challenge by default.

Extracting the `init` response data

You need to extract the `WL-Instance-Id` and the `token` from the `init` response and send them as headers in all requests to the Worklight Server. If you do not do so, the authentication check fails and the request is rejected. Challenge data is different for each session, so you need to extract and store the challenge data for each thread. For more information, see "Testing back-end invocation" on page 188 later in this section.

Changing the response status to HTTP 200

When the performance testing thread runs the initialization for the first time, Worklight Server responds to challenge data with an HTTP 401 status. This is to be expected, so the performance tool should treat this HTTP status as a success. The HTTP status can be changed to HTTP 200 by using the performance testing tool's script. In this way, the performance testing tool will record the request as a success, otherwise the performance testing report might mark this request as having failed

and might record it as an error. This would greatly impact the performance testing report.

Testing back-end invocation

You should start testing back-end invocation only after you have finished testing authentication flow. You can choose any type of back end that you want. The request for the back-end invocation has the following structure:
`http://{Host}:{Port}/{Context}/apps/services/api/{AppName}/{environment}/query.`

Table 47. Backend invocation parameters

Parameter	Description
adapter	Worklight Adapter name.
procedure	Worklight procedure name
parameters	Procedure parameters should be an array.

The following figure shows an example array of parameters:

The following figure shows an example of request headers:

By default, the jMeter tool encodes the URL. If your performance testing tool does not support URL encoding, you must use encoded parameter values.

For the iPhone and Android environments, since they contain `wl_deviceNoProvisioningRealm` by default, you need to send the Authorization header. The format for HTTP Authorization header is shown as follows. You need to replace *${device-token}* with the token you extracted in the initialization phase. `{"wl_deviceNoProvisioningRealm":{"device":{"id":"1234567890","os":"5.0","model":"testModel` When the response data `"isSuccessful"` is true, this indicates that the response data from the Worklight Adapter procedure was successfully received and now you can continue with your back-end testing.

Logging in

When the Worklight adapter procedure is protected by a security test or the `connectAs` property is set to `endUser`, you need to log in to the Worklight Server before calling this procedure. To check if the Worklight adapter procedure needs a login, you can call the procedure followed by the steps described earlier, and check the response data from the Worklight Server. If the response data includes `isSuccessful:true` and `authStatus:required`, you should log in to the Worklight Server first, otherwise the requests to this procedure are rejected by Worklight Server. The way you log in to the Worklight Server depends on the authentication type. If the app is protected by form-based authentication or adapter-based authentication, you can call the login procedure after successfully completing initialization. In general, the login procedure should not be protected by a security test; it can be directly called after initialization is completed. For other authentication types, you can capture the network traffic on Worklight Server by

using network traffic capture tools (for example, Fiddler or Wireshark). The network traffic data shows the detailed URL and parameters that you can use to log in to the Worklight Server. The following screen image shows an example of a login function that calls the `setActiveUser` API with the supplied user ID and password:

Logging out

The following options are available for logging out of the Worklight Server:

Not logging out for each iteration
> Worklight Server automatically logs the user out when the session times out. This option consumes more memory than logging out, but is useful if you want to maximize memory usage during performance testing. To adopt this option, you need to clean cookies for each iteration in the performance testing tool.

Logging out after each iteration by using the Worklight logout service
> It is recommended to clean cookies for each iteration to avoid sharing data between iterations. The logout request has the following structure:
> `http://{Host}:{Port}/{Context}/apps/services/api/{AppName}/{environment}/logout`

For more information about the parameters, see "HTTP Interface of the production server" on page 1011.

Database reporting

To activate database reporting, you need to specify `reports.exportRowData=true` in your `worklight.properties` file. You also need to set up the reports database. For more information, see "Reports database" on page 1423. After you enable database reporting, you can use the back-end invocation step described earlier. See the database reporting section in the following document: Scalability and Hardware Sizing.

Single sign-on (SSO), direct update, push notification, and geolocation

See the relevant section in the following document: Scalability and Hardware Sizing.

General example: Using jMeter as a performance testing tool

HTTP cookie management

> Cleaning the cookies on every thread iteration ensures that no data and user information is being cached during this iteration. If you want to keep cookie information, you need to clean the user information at the end of the iteration to avoid unexpected errors during load testing. For example, if the user does not log out during the previous iteration, the next iteration might be affected by that user.

HTTP Header Management

> The necessary `x-wl-platform-version` and `x-wl-app-version` that were described earlier can be defined here; you can also define the

WL-Instance-Id and WL_deviceNoProvisioningRealm token placeholders. You can use a jMeter script to extract the real challenge data and replace the placeholders for each thread iteration as shown in the following image:

Initialization phase

1. Extract and replace the `WL-Instance-Id` placeholder:

2. Extract and replace the `WL_deviceNoProvisioningRealm` token placeholder:

3. Change initialization HTTP status 401 and 403 to HTTP status 200:

Security configuration

Configure the security of the Worklight Server as detailed here.

Securing the Worklight Server administration

This section helps ensure that no unauthorized person can perform Worklight Server administration operations. This is particularly important in a production environment.

The security threat is that any person who can install mobile applications in a production environment is able to modify the behavior of these apps on the mobile devices. The apps are served to the clients through the Worklight runtime environments, which get these apps from the Administration Services through JMX. The Administration Services fetch these apps from the administration database. The Administration Services and the Worklight Console allow any user in the roles of `worklightadmin` or `worklightdeployer` to deploy applications. A similar threat exists for adapters.

Enabling https in the application server

The ability to use `https` with the application server is a prerequisite.

For WebSphere Application Server Liberty profile:
*   Verify that the `server.xml` file contains `<feature>ssl-1.0</feature>`. At minimum, the file must contain `<feature>restConnector-1.0</feature>` because the `restConnector-1.0` feature implies that the `ssl-1.0` feature is enabled.
*   Verify that the HTTPS port is enabled, by ensuring that this HTTPS port does not have an `<httpEndpoint>` element with a `httpsPort` attribute that is negative. If the HTTPS port is disabled, SSL is also disabled, and the JMX connections that the Worklight Server requires do not work.
*   Verify that the `server.xml` file contains `<keyStore id="defaultKeyStore" .../>`, or an equivalent declaration, otherwise the JMX connections that the Worklight Server requires do not work. For more information, see Liberty profile: SSL configuration attributes.

For Apache Tomcat:
*   Enable an `https` port as documented in SSL support and SSL Configuration HOW-TO.

Enabling application security in the application server

Without this step, anyone can connect to the web applications without credentials.

For WebSphere Application Server full profile:
* Verify that Administrative Security is enabled.
* Verify that Application Security is enabled.

For WebSphere Application Server Liberty profile:
* Verify that the `server.xml` file contains `<feature>appsecurity-1.0</feature>`.

Protecting the passwords of users in the roles `worklightadmin` and `worklightdeployer`

If the password of any user who is mapped to the roles `worklightadmin` or `worklightdeployer` is compromised, that is, becomes potentially known to an unauthorized person, unauthorized Worklight administration operations are possible. Here are steps to mitigate this risk.

* Minimize the number of users that you map to the roles `worklightadmin` and `worklightdeployer`.
* Map different users to the roles `worklightadmin` or `worklightdeployer` in development and test environments than you do in the production environment. If the password of the administrator of the development or test environment is compromised (for example, by use of `secure="false"`), this helps secure the password of the administrator of the production environment.
* If these users are authenticated through LDAP, secure the connection to the LDAP server.
* Never use the Worklight Console or the Worklight Administration REST services over `http`. Always use `https`. There are two ways to guarantee this:
    – Configure the application server to respond only to an `https` port, not to an `http` port.
    – Modify the `worklightconsole.war` and `worklightadmin.war` files to activate the JEE6 transport security of type `CONFIDENTIAL`. This setting performs a redirect from `http` to `https` before the application server requests a user and password.
        1. Unpack `worklightconsole.war` (as a `.zip` file).
        2. Edit its `WEB-INF/web.xml file`, changing `<transport-guarantee>NONE</transport-guarantee>` to `<transport-guarantee>CONFIDENTIAL</transport-guarantee>`.
        3. Repack `worklightconsole.war`.
        4. Unpack `worklightadmin.war` (as a `.zip` file).
        5. Edit its `WEB-INF/web.xml file`, changing `<transport-guarantee>NONE</transport-guarantee>` to `<transport-guarantee>CONFIDENTIAL</transport-guarantee>`.
        6. Repack `worklightadmin.war`.
        7. Redeploy these WAR files, either manually, or through the Ant task `<installworklightadmin>` or `<updateworklightadmin>`. For more information, see "Deploying the Worklight Console and Administration Services with Ant tasks" on page 117.
* Never use the `<wladm>` Ant task with the attribute `secure="false"`, and never use the `wladm` command with the option `-secure=false`. To achieve this, you must:

- Ensure that your application server uses an SSL certificate signed by a CA, not a self-signed certificate, and that the host name mentioned in this certificate matches the host name of the application server machine.
  - Ensure that this SSL certificate is contained in the truststore of the JVM that runs the `<wladm>` Ant task or the `wladm` command.
- Change the file access permissions of the file that contains the password that is used by the `<wladm>` Ant task or the `wladm` command to be as restrictive as possible. To do this, you can use a command, such as the following examples:
  - On UNIX: `chmod 600 adminpassword.txt`
  - On Windows: `cacls adminpassword.txt /P Administrators:F %USERDOMAIN%\%USERNAME%:F`
- Additionally, you might want to obfuscate the password, to hide it from an occasional glimpse. To do so, use the `wladm config password` command to store the obfuscated password in a configuration file. Then you can copy and paste the obfuscated password to the Ant script or to the password file that you want.
- In the configuration of the Worklight Console web application, set the JNDI property `ibm.worklight.admin.ui.cors.strictssl` to `true`. This property helps rejecting unsecure SSL certificates.
- In the configuration of the Worklight Console web application, set the JNDI property `ibm.worklight.admin.hsts` to `true`. This property implements HTTP Strict Transport Security and helps the administrator's browser remember to access the Worklight Console through `https` instead of `http`.

Protecting the administration database

If the password of the administration database (or of the user who owns the corresponding schema of that database) is compromised, that is, becomes potentially known to an unauthorized person, unauthorized deployments of apps and adapters are possible. Here are steps to mitigate this risk.

- Do not host other services than the database management system on the machines that serve this database.
- If you use Ant tasks to configure the Worklight Server administration (see "Using Ant tasks to install Worklight Server administration" on page 113), you must do one of the following actions:
  - Change the file access rights of the Ant XML file to be as restrictive as possible before you store passwords in it. For more information, see step 2 in "Sample configuration files" on page 1156.
  - Write ************ (12 asterisks) in place of the password, so the Ant XML file does not contain the password. Instead, the Ant task queries the password interactively when it is invoked.
- Minimize the number of users who have login access to the machines that run Worklight Server.
- Change the file access rights of the application server configuration files that contain the `jdbc/WorklightAdminDS` data source password to be as restrictive as possible. For more information, see step 3 in "Sample configuration files" on page 1156.

Protecting the JMX communication

If the JMX communication between Administration Services and the Worklight runtime environments are not secured, unauthorized persons who have local access to the Worklight Server machines can play man-in-the-middle attacks and thus activate tampered apps and adapters. Here are steps to mitigate this risk.

- For WebSphere Application Server Liberty Profile, follow the procedure of Configuring secure JMX connection to the Liberty profile.
- For Apache Tomcat, use a JMX configuration with SSL, as described in "Configuring Apache Tomcat" on page 108.

Protecting the apps and adapters to deploy

If the source from which the Worklight administrator receives apps and adapters is not secured, tampered apps and adapters can be submitted to the Worklight administrator, who then deploys them. Here are steps to mitigate this risk.

- Ensure that the Worklight administrator receives apps and adapters only through channels which guarantee the integrity of the sender and of the sent artifacts. For example, use emails with digital signature, or web-based tools with the need to log in through `https`.
- Ensure that the development teams that create these apps and adapters use a Version Control System that guarantees the integrity of each modification and disallows modifications by unauthorized persons. Examples of VCS systems in this category are RTC/jazz and Git. An example of a VCS not in this category is CVS.

Protecting against attacks from the internet

Attackers from the internet might attempt to search for security flaws in the Worklight Console and Administration Services and try to circumvent the security measures. Here is a tip to mitigate this risk. It assumes that mobile application users connect to Worklight Server from the internet, but all legitimate uses of the Worklight Console and Administration Services are from an intranet.

- Configure your internet gateway or firewall (for example, IBM DataPower®) to block access to URLs under the context roots of the Worklight Console (default: `/worklightconsole`) and of the Administration Services (default: `/worklightadmin`). At the same time, keep the access to the Worklight runtime web applications open.

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 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 Worklight applications on the server" on page 1171 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 1178.

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.

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.

Configuring WebSphere Application Server Liberty profile when IBM JDK is used

When you configure the Worklight Server on WebSphere Application Server Liberty profile and you use the IBM JDK, you must use default JSSE socket factories instead of the SSL socket factories of WebSphere Application Server.

You must follow the same instructions as "Configuring Liberty profile when IBM JDK is used" on page 269 to apply the same configuration to the Worklight Server.

Transmitting Worklight data on the BlackBerry Enterprise Server MDS channel

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

About this task

Figure 8 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 Foundation in environments such as these, you can configure Worklight data to use the same channel.
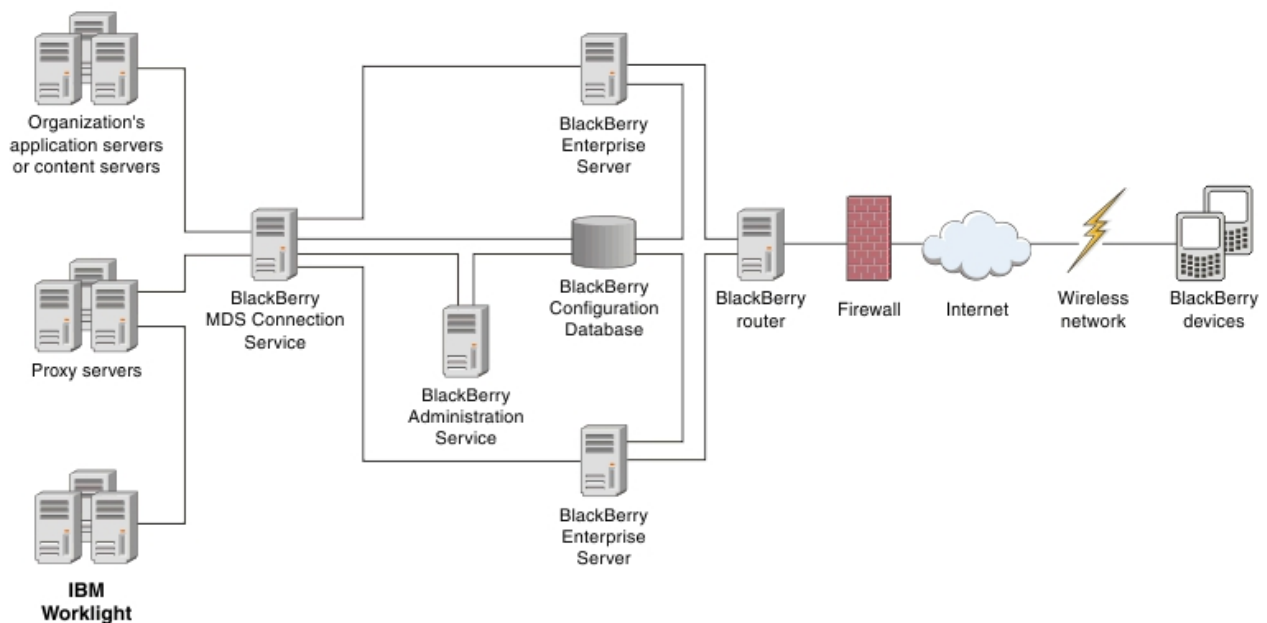
Figure 8. Worklight with BlackBerry Enterprise Server

Procedure

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

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

Procedure

1. Set up Worklight-specific configuration.

   a. For each app that you are configuring, modify the `authenticationConfig.xml` file on the server to include the following security test, realm, and login module declarations:

   ```
   <securityTests>
     <mobileSecurityTest name="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.submitLoginFormCallb

                      });
```

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.isUserAuthentica
    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 tag in the project `application-descriptor.xml` file. Set the value of this attribute to the name of the security test that you declared in the `authenticationConfig.xml` file in step 1a. Here is an iPad example:

```
<ipad bundleId="com.Datapower" securityTest="WASTest-securityTest" version="1.0">
  <worklightSettings include="false"/>
  <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 48. 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 50 on page 200.)
	
	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.

Table 48. General Configuration (continued)

Field	Description
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 201. • Policy `mpgw-form` for HTML form-based login authentication: see "Rules for HTML form-based authentication" on page 203.
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 49. 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 50. 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.

Table 50. AAA policy for HTML forms-based authentication (continued)

Phase	Description
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 51. 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**
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 that is named `worklight-basicauth`.

You create the `worklight-basicauth` policy as part of the process of defining a multiprotocol gateway. See "Protecting your mobile application traffic by using IBM WebSphere DataPower as a security gateway" on page 196, Table 48 on page 199.

Table 52. HTTP Basic Authentication properties

Property	Value
Policy Name	`worklight-basicauth`

Table 52. HTTP Basic Authentication properties (continued)

Property	Value
Order of configured rules	1. `worklight-basicauth_rule_0`: see Table 53 2. `worklight-basicauth_rule_3`: see Table 56 on page 203 3. `worklight-basicauth_rule_1`: see Table 54 4. `worklight-basicauth_rule_2`: see Table 55

Table 53. Properties of `worklight-basicauth_rule_0`. When processing HTML content, skip processing with the icon that is associated with the website or the web page.

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 54. Properties of `worklight-basicauth_rule_1`. Handle end-user authentication if an LTPA token does not exist.

Property	Value
Direction	**Client to Server**.
Match	• Type = URL • Pattern = *
AAA	BasicAuth2LTPA • Output: NULL
Result	Not applicable.

Table 55. Properties of `worklight-basicauth_rule_2`. Handle both the redirect and content-type reset on the response side.

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 206. • Output: NULL
Result	Not applicable.

Table 56. Properties of `worklight-basicauth_rule_3`. Because the policy is applied to each request, the rules must be ordered such as to ensure that an LTPA token is verified if it exists in the HTTP request. If no token is available, proceed to the next rule and authenticate the user.

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 form-based authentication

Add rules to define an HTML form-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 196, Table 48 on page 199.

Table 57. HTTP Form-Based Login properties

Property	Value
Policy Name	`mpgw-form`
Order of configured rules	1. `mpgw-form_rule_0`: see Table 58 2. `mpgw-form_rule_1`: see Table 59 3. `mpgw-form_rule_2`: see Table 60 on page 204 4. `mpgw-form_rule_3`: see Table 61 on page 204 5. `mpgw-form_rule_6`: see Table 62 on page 204

Table 58. Properties of `mpgw-form_rule_0`. This rule skips processing with the icon that is associated with the web site or the web page.

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 59. Properties of `mpgw-form_rule_1`. This rule verifies an LTPA token if it exists in the HTTP request.

Property	Value
Direction	**Client to Server**.

Table 59. Properties of `mpgw-form_rule_1` *(continued).* This rule verifies an LTPA token if it exists in the HTTP request.

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

Table 60. Properties of `mpgw-form_rule_2`. This rule generates the HTML form login page.

Property	Value
Direction	**Client to Server**.
Match	• Match with PCRE = on • Type = URL • Pattern = `/(Login|Error)Page\.htm(1)?(\ ?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 205. **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 61. Properties of `mpgw-form_rule_3`. This rule handles end-user authentication if an LTPA token does not exist.

Property	Value
Direction	**Client to Server**.
Match	• Type = URL • Pattern = `*`
Advanced	"Convert Query Parameter to XML". Accept default values for other selections.
AAA	Form2LTPA

Table 62. Properties of `mpgw-form_rule_6`. This rule handles both the redirect and content-type reset on the response side.

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

Table 62. Properties of `mpgw-form_rule_6` *(continued).* This rule handles both the redirect and content-type reset on the response side.

Property	Value
Filter	Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see "Sample redirect stylesheet" on page 206. • 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 form-based authentication" on page 203, Table 60 on page 204.

```
<?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 form-based authentication" on page 203, Table 62 on page 204),
and when you define rule `worklight-basicauth_rule_2` (see "Rules for HTTP basic
authentication" on page 201, Table 55 on page 202).

```
<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 Worklight Server to enable TLS V1.2

For Worklight Server to communicate with devices that support only TLS V1.2,
among the SSL protocols, you must complete the following instructions.

About this task

The steps to configure Worklight Server to enable Transport Layer Security (TLS)
V1.2 depend on how Worklight Server connects to devices.

If Worklight Server is behind a reverse proxy that decrypts SSL-encoded packets from devices before passing the packets to the application server, you must enable TLS V1.2 support on your reverse proxy. If you are using IBM HTTP Server as your reverse proxy, see Securing IBM HTTP Server for instructions.

If Worklight Server communicates directly with devices, the steps to configure Worklight Server to enable TLS V1.2 depend on the application server that you use. The following sections provide you with the specific instructions for Apache Tomcat, WebSphere Application Server Liberty profile, and WebSphere Application Server full profile.

Apache Tomcat
Procedure
1. Confirm that the Java Runtime Environment (JRE) supports TLS V1.2.

   Ensure that you use Oracle JRE 1.7.0_75 or later for Worklight Server V6.2.
2. Edit the `conf/server.xml` file and modify the `<Connector>` element that declares the HTTPS port so that the `sslEnabledProtocols` attribute has the following value:

   ```
   sslEnabledProtocols="TLSv1.2,TLSv1.1,TLSv1,SSLv2Hello"
   ```

WebSphere Application Server Liberty profile
Procedure
1. Confirm that the Java Runtime Environment (JRE) supports TLS V1.2.
   * If you use an IBM Java SDK, ensure that your IBM Java SDK is patched for the POODLE vulnerability. You can find the minimum IBM Java SDK versions that contain the patch for your version of WebSphere Application Server in Security Bulletin: Vulnerability in SSLv3 affects IBM WebSphere Application Server (CVE-2014-3566).

     Note: You can use the versions that are listed in the security bulletin or later versions.
   * If you use an Oracle Java SDK, ensure that you use Oracle JRE 1.7.0_75 or later.
2. If you use an IBM Java SDK, edit the `server.xml` file.
   a. Add the following line:
      ```
      <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="SSL_TLSv2"/>
      ```
   b. Add the `sslProtocol="SSL_TLSv2"` attribute to all existing `<ssl>` elements.

WebSphere Application Server full profile
Procedure
1. Confirm that the Java Runtime Environment (JRE) supports TLS V1.2.

   Ensure that your IBM Java SDK is patched for the POODLE vulnerability. You can find the minimum IBM Java SDK versions that contain the patch for your version of WebSphere Application Server in Security Bulletin: Vulnerability in SSLv3 affects IBM WebSphere Application Server (CVE-2014-3566).

   Note: You can use the versions that are listed in the security bulletin or later versions.
2. Log in to WebSphere Application Server administrative console, and click **Security** > **SSL certificate and key management** > **SSL configurations**.
3. For each SSL configuration listed, modify the configuration to enable TLS V1.2.

a. Select an SSL configuration and then, under **Additional Properties**, click **Quality of protections (QoP) settings**.

b. From the **Protocol** list, select SSL_TLSv2.

c. Click **Apply** and then save the changes.

Configuring SSL between Worklight adapters and back-end servers by using self-signed certificates

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

Procedure

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

```
##########################################################################
#     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 that the keystore file exists in the `server/conf` folder of the Worklight project.

3. Export the server 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 in 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 must 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 must 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 must 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
   ```

   Note: The **First and Last Name** field contains your server URL, which you use in the `adapter.xml` configuration file, 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 certi
    <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 by using untrusted certificates

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

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

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

Certificate authority (CA)

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 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.

Self-signed certificate

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

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

Self-signed CA

A CA that 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.

Certificate types that are supported by different mobile platforms

Table 63. 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	-	⌂	⌂	⌂
Windows Phone	-	⌂	⌂	⌂
Blackberry	-	⌂	⌂	⌂
Windows 8	-	⌂	⌂	⌂

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 truststore. This restriction makes it impossible for the client to ever trust the server's certificate. Although self-signed certificates are often recommended for development and testing purposes, they will not work when the client is a mobile device.

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

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

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

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 216.

For Android, see "Installing the root CA on Android" on page 219.

For Windows Phone, see "Installing the root CA on Windows Phone" on page 220.

For Windows 8, see "Installing the root CA on Windows 8" on page 224.

Configuring Android

It is important to note that if the following flag is set to `true` in your application, Android ignores SSL errors under certain conditions:

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

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 228.

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 64. 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 211. Ensure that the certificate is in PEM format. Ensure that the certificate has a `.crt` file extension.
Unable to install the root CA on Android. After installation, the certificate does not show up in the system 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 211. Ensure that the certificate is in PEM or DER format. Ensure that the certificate has a `.crt` file extension.

Table 64. Troubleshoot problems with server certificates (continued)

Problem	Actions to take
`"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 212. 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 213. 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 213. 2. Ensure that the CRL URL is valid and reachable. For more information, see "CRL support" on page 213. 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 213.
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. Verify that the `debuggable` flag is set to `false` (debuggable:false) in the Android manifest file, 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 configuring Android, see "Configuring Android" on page 212.

Table 64. Troubleshoot problems with server certificates (continued)

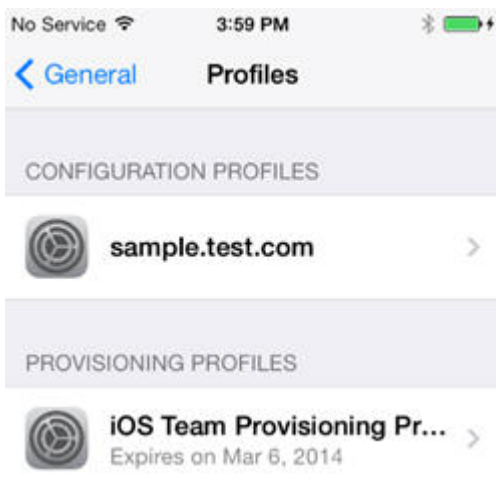| Problem	Actions to take
After installation, the certificate does not show up in the system's trusted credentials or truststore.	Ensure that you did not install the server certificate by accessing the protected resource directly from your browser. This action imports the certificate only into the browser 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 219.
`SCRIPT7002: XMLHttpRequest: Network Error 0x2ee4, Could not complete the operation due to error 00002ee4`	• Ensure that you installed the server's root CA on the client device. For more information, see "Establishing trust on the client" on page 212. • Ensure that the server sends the complete certificate chain and in the right order. For more information, see "Handling the certificate chain" on page 213.

Related tasks:

"Configuring SSL for Liberty profile" on page 296
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.

No Service 🛜   3:59 PM   ⚡ 🔋 ⚡

Cancel   **Warning**   Install

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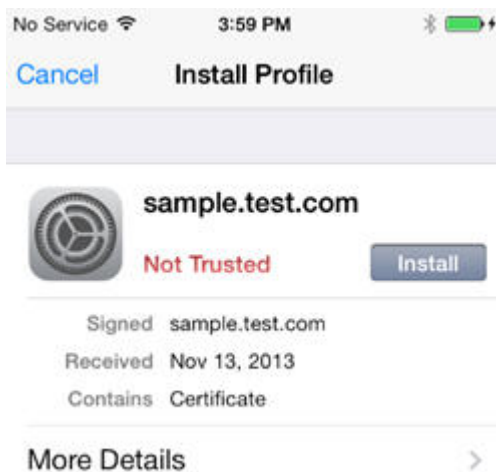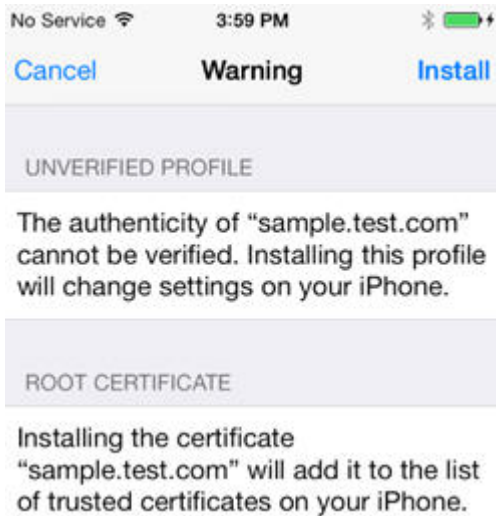
7. Check that the certificate was properly installed under **Settings** > **General** > **Profiles** > **Configuration Profiles**.

No Service 🛜   3:59 PM   ⚡ 🔋 ⚡

❮ General   **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   ›
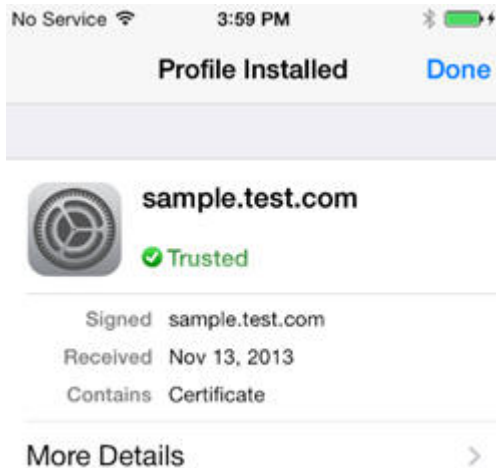
Installing the root CA on Android

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`

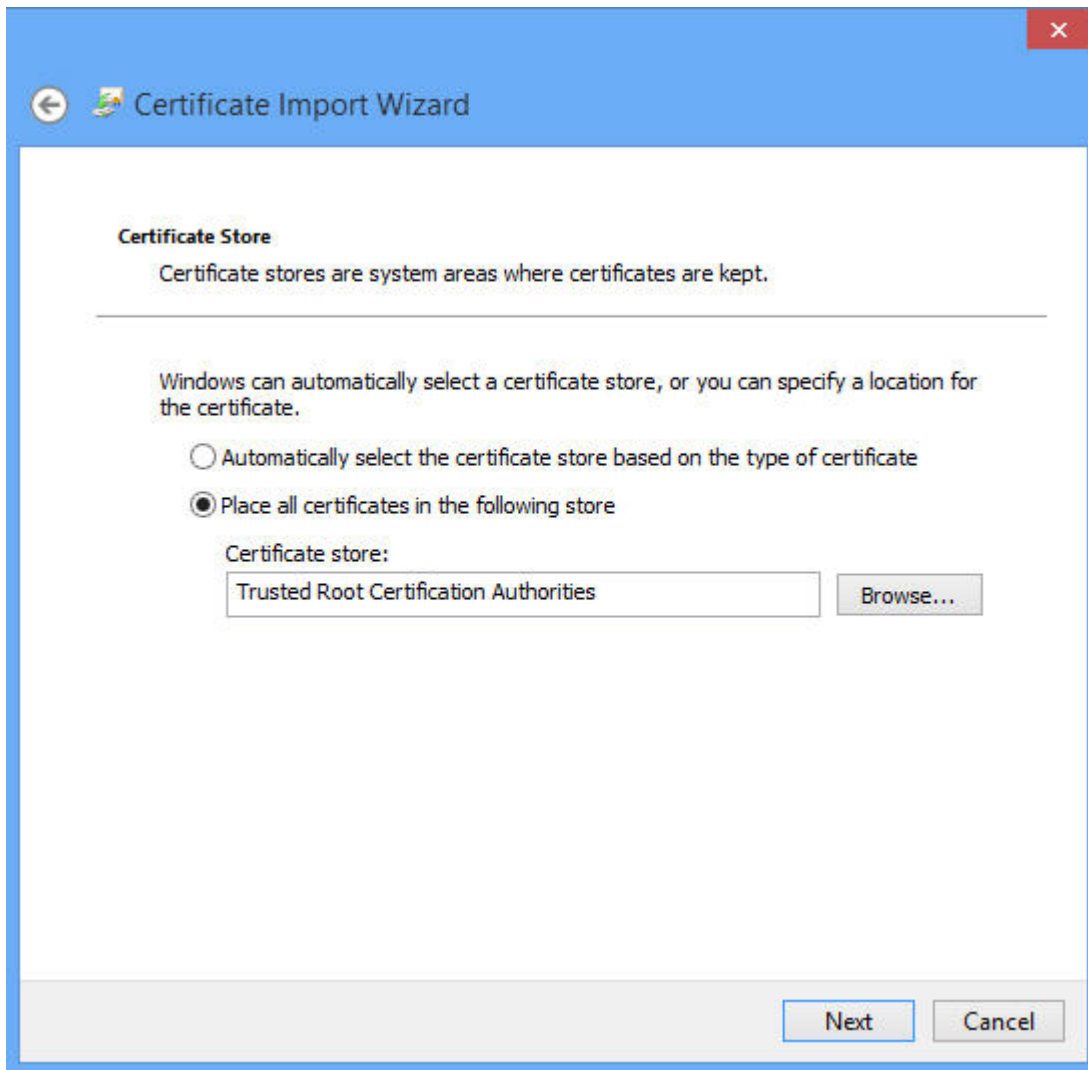   **Note:** The following `openssl` flag generates the CA extension:

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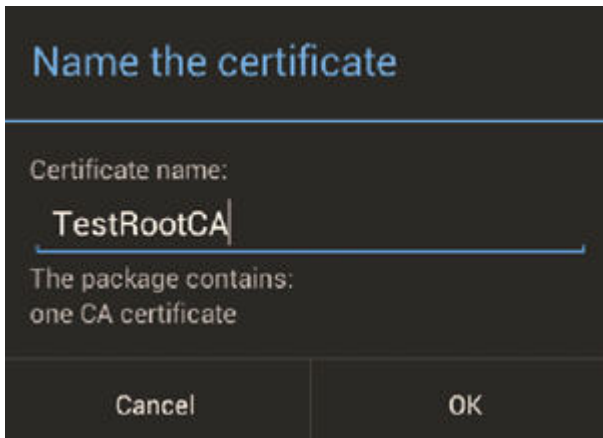
8. Check that the certificate was properly installed under **Settings** > **Security** > **Trusted Credentials** > **User**.

Installing the root CA on Windows Phone

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 `DER` file format and has a `.cer` file extension. The PEM with a `.crt` file extension is not supported. Convert as needed.

2. Run the following command to view the certificate details.

   `openssl x509 -inform DER -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.

6. Click the link in your email or on the website, and then click **Tap to open** and confirm the installation.

Tap to open
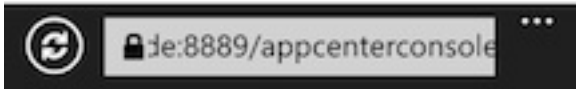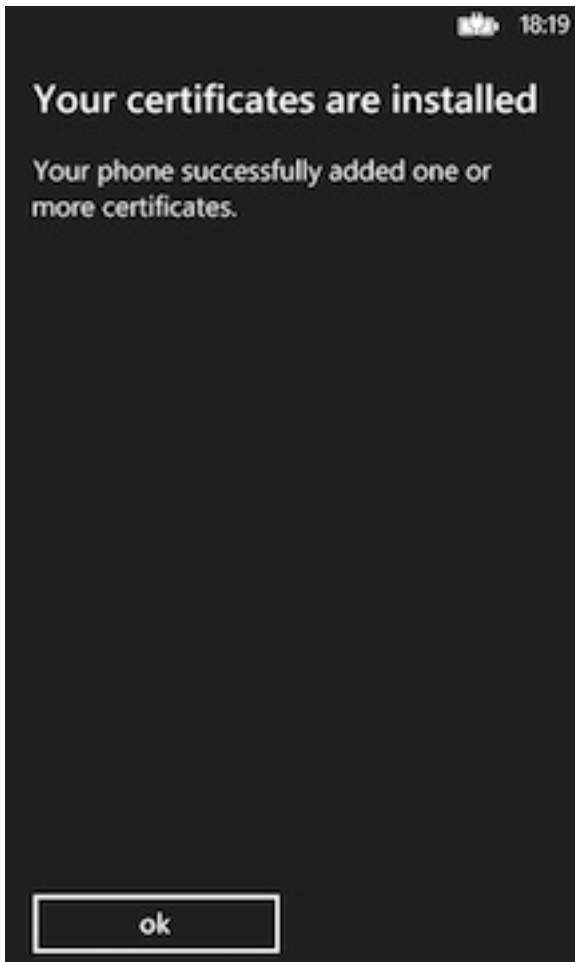
Results

You can now use web servers that are secured with certificates that are based on this root CA.

Note: There is no way on Windows Phone to check whether the certificate was properly installed. Furthermore, after you install the certificate, Windows Phone provides no way to remove the certificate from the device.

Installing the root CA on Windows 8

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 DER file format and has a `.cer` file extension. The PEM with a `.crt` file extension is not supported. Convert as needed.

2. Run the following command to view the certificate details.

```
openssl x509 -inform DER -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.

6. Click the link in your email or on the website, and on the General tab in the certificate window, select **Install Certificate** and click **OK** to confirm the installation.
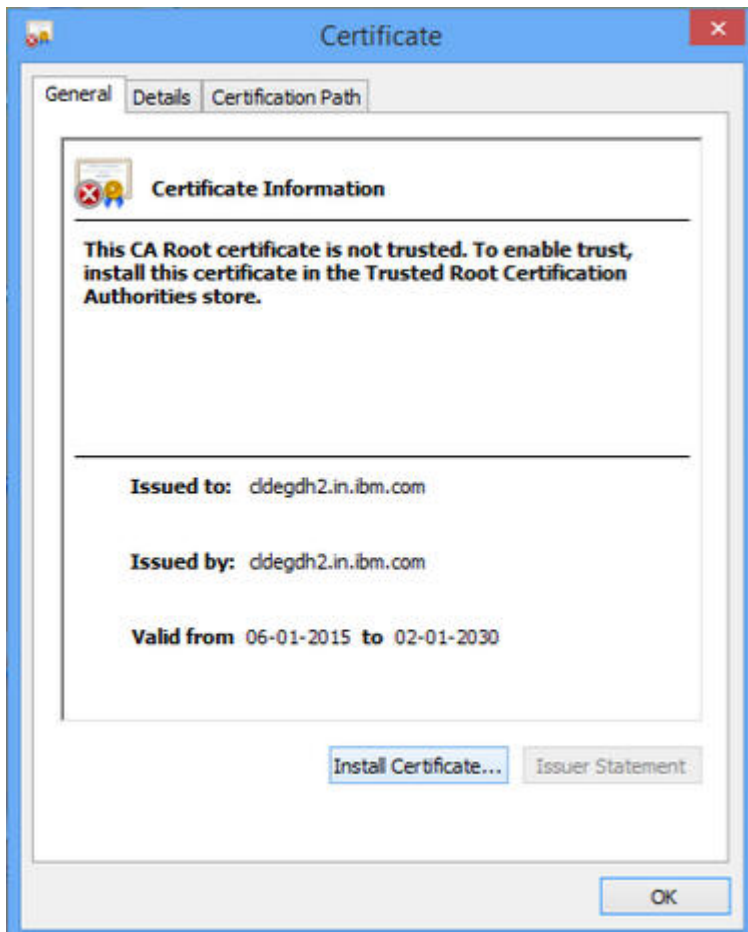
7. In the Certificate Import Wizard welcome screen, select **Local Machine** as the Store Location and click **Next**.

8. Choose **Place all certificates in the following store** option and select **Trusted Root Certification Authorities** as the certificate store. Click **Next**.

Certificate Import Wizard

Certificate Store
Certificate stores are system areas where certificates are kept.

Windows can automatically select a certificate store, or you can specify a location for the certificate.

○ Automatically select the certificate store based on the type of certificate

◉ Place all certificates in the following store

Certificate store:

Trusted Root Certification Authorities   Browse...

Next   Cancel

9. Choose **Certificate Store Selected by User** and click **Finish**.
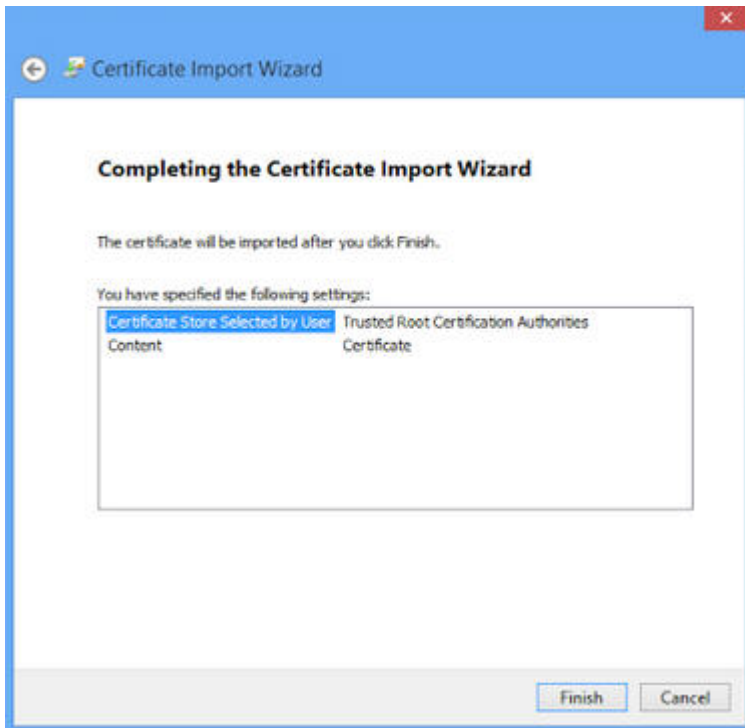
10. You will receive a **Import successful** message, select **OK**. The certificate is now installed.

Results

You can now use web servers that are secured with certificates that are based on this root CA.

Note: There is no way on Windows 8 to check whether the certificate was properly installed. Furthermore, after you install the certificate, Windows 8 provides no method to remove the certificate from the device.

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 INTERMEDIATE_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.

WebSphere Application Server SSL configuration when using HTTP adapters

By setting a property, you can make HTTP adapters take benefit of the WebSphere SSL configuration.

By default, HTTP adapters do not take benefit of the WebSphere SSL configuration by concatenating the Java Runtime Environment (JRE) truststore with the Worklight truststore as referenced by the **ssl.keystore.path**, **ssl.keystore.password**, and **ssl.keystore.type** properties. See "Configuring SSL between Worklight adapters and back-end servers by using self-signed certificates" on page 209. To have HTTP adapters use the WebSphere SSL configuration, set the **ssl.websphere.config** property to true. This value has the following effects, in order of precedence:

1. If the **ssl.keystore.path**, **ssl.keystore.password**, **ssl.keystore.type** properties are set, the adapter uses the truststore that is referenced in these properties without concatenating it with the JRE truststore.
2. If the **ssl.websphere.alias** property is set, the adapter uses the SSL configuration that is associated with the alias set in this property.
3. If the **ssl.keystore.path**, **ssl.keystore.password**, **ssl.keystore.type**, and **ssl.websphere.alias** properties are not set, the WebSphere outbound dynamic configuration is used.

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 st
```

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.

Managing the DB2 transaction log size

When you deploy an application that is at least 40 MB with IBM Worklight Console, you might receive a transaction log full error.

About this task

The following system output is an example of the transaction log full error code.

DB2 SQL Error: SQLCODE=-964, SQLSTATE=57011

The content of each application is stored in the IBM Worklight administration database.

The active log files are defined in number by the **LOGPRIMARY** and **LOGSECOND** database configuration parameters, and in size by the **LOGFILSIZ** database configuration parameter. A single transaction cannot use more log space than **LOGFILSZ** * (**LOGPRIMARY** + **LOGSECOND**) * 4096 KB.

The **DB2 GET DATABASE CONFIGURATION** command includes information about the log file size, and the number of primary and secondary log files.

Depending on the largest size of the IBM Worklight application that is deployed, you might need to increase the DB2 log space.

Procedure

Using the **DB2 update db cfg** command, increase the **LOGSECOND** parameter. Space is not allocated when the database is activated. Instead, the space is allocated only as needed.

Installing the IBM Worklight Analytics Platform

The IBM Worklight Analytics Platform is delivered as a WAR file. When you develop within Worklight Studio, the WAR file that contains the IBM Worklight Analytics Platform is automatically deployed. The Worklight Server forwards data to the IBM Worklight Analytics Platform with no additional required configurations.

The analytics WAR file is included with the Worklight Server installation. For more information, see "Distribution structure of Worklight Server" on page 100.

The following sections describe the required steps for successfully deploying the WAR file to the application server.

When you deploy the WAR file, the analytics console is available at:

```
http://<hostname>:<port>/<context-root>/console
```

Example:

```
http://localhost:9080/analytics/console
```

The following procedures are used for installing a single node on an application server. For more information about configuring and installing a cluster, see "Production deployment and clustering" on page 1402.

Installing IBM Worklight Analytics Platform for WebSphere Application Server Liberty

You can install the IBM Worklight Analytics Platform for WebSphere Application Server Liberty.

About this task

The following steps describe how to install and run the Analytics WAR file on WebSphere Application Server Liberty

Procedure

1. Add the Analytics WAR file to the `apps` folder of your WebSphere Application Server Liberty application server.
2. Modify the `server.xml` file to set the context root of the WAR file.

   ```
   <application context-root="/analytics"
                id="analytics"
                location="analytics.war"
                name="analytics"
                type="war">
     <classloader delegation="parentLast"/>
   </application>
   ```
3. Add the following features to the WebSphere Application Server Liberty server in the feature manager.

   ```
   <feature>jsp-2.2</feature>
   <feature>jndi-1.0</feature>
   <feature>appSecurity-1.0</feature>
   ```
4. Start the application server.

   ```
   http://localhost:9080/analytics/console
   ```

Results

The analytics console is deployed and can now be viewed in the browser.

Installing IBM Worklight Analytics Platform for WebSphere Application Server

You can install the IBM Worklight Analytics Platform for WebSphere Application Server.

About this task

The following steps describe how to install and run the Analytics WAR file on WebSphere Application Server

Procedure

1. Deploy the WAR file to the application server, but do not start it.
2. Select the application from the **Enterprise Applications** list and click on **Class loading and update detection**.

3. Set the classloading order to **parent last**.

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

[Apply] [OK] [Reset] [Cancel]

4. Click on **Manage Modules**.

5. Select the module and change the class loading order to **parent last**.

6. Start the application and visit the link in the browser.

Results

The analytics WAR file is now ready to accept incoming analytics data.

IBM Worklight Analytics Platform installation for Tomcat

Follow the normal procedures for deploying a WAR file. No special configurations need to be made for Tomcat.

Configuring the Worklight Server for the IBM Worklight Analytics Platform

You must configure the Worklight Server for the IBM Worklight Analytics Platform.

About this task

The following steps describe how to configure the Worklight Server for the IBM Worklight Analytics Platform.

Procedure

In the `worklight.properties` file, set the `wl.analytics.url` property to point to the deployed WAR file.

`wl.analytics.url=http://<hostname>:<port>/<context-root>/data`

For example, if the Liberty server is at `host.ibm.com` on port 8080 and the context root is `analytics`, then the `wl.analytics.url` property is as follows:

`wl.analytics.url=http://host.ibm.com:8080/analytics/data`

Results

The Worklight Server now forwards data to the IBM Worklight Analytics Platform.

Note: All properties in the `worklight.properties` file can also be set by using JNDI. For more information about JNDI settings, see "Configuration of Worklight applications on the server" on page 1171.

Installing and configuring the Application Center

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

The Application Center is part of Worklight Server. To install the Application Center, see the following topics. Optionally, you can install the database of your choice before you install Worklight Server with the Application Center.

When you install an IBM Worklight Foundation 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. For more information, see "Configuring the Application Center after installation" on page 264.

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

If you intend to install applications on iOS devices through the Application Center, you must first configure the Application Center server with SSL.

For a list of installed files and tools, see "Distribution structure of Worklight Server" on page 100.

Installing Application Center with IBM Installation Manager

With IBM Installation Manager, you can install Application Center, create its database, and deploy it on an Application Server.

Before you begin

Verify that the user who runs IBM Installation Manager has the privileges that are described in "File system prerequisites" on page 72.

Procedure

To install IBM Application Center with IBM Installation Manager, complete the followings steps.

1. Optional: You can manually create databases for Application Center, as described in "Optional creation of databases." IBM Installation Manager can create the Application Center databases for you with default settings.
2. Run IBM Installation Manager, as described in "Running IBM Installation Manager" on page 84.
3. Select **Yes** to the question **Install IBM Application Center**.

Optional creation of databases

If you want to activate the option to install the Application Center when you run the 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 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 Foundation 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 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 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 Foundation runs.

Creating the Oracle database for Application Center:

During the 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 system privilege: **CREATE SESSION**
        - Add system privilege: **CREATE SEQUENCE**
        - Add system privilege: **CREATE TABLE**
        - 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 CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO APPCENTER;
     DISCONNECT;
     ```

Installing Worklight Server in WebSphere Application Server Network Deployment

To install Worklight Server in 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 Foundation 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 65. Selecting the required scope

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

4. Restart the target servers by following the procedure in "Completing the installation."

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

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 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 Application Center is installed in the application server. A Worklight Console is not installed by default. To install a Worklight Console, you need to follow the steps in Chapter 12, "Deploying Worklight projects," on page 1101.

Default logins and passwords created by IBM Installation Manager for the Application Center

IBM Installation Manager creates the logins by default for the Application Center, according to your application server. You can use these logins to test the Application Center.

WebSphere Application Server full profile

The login `appcenteradmin` is created with a password that is generated and displayed during the installation.

All users authenticated in the application realm are also authorized to access the `appcenteradmin` role. This is not meant for a production environment, especially if WebSphere Application Server is configured with a single security domain.

For more information about how to modify these logins, see "Configuring WebSphere Application Server full profile" on page 265.

WebSphere Application Server Liberty profile

- The login `demo` is created in the `basicRegistry` with the password `demo`.
- The login `appcenteradmin` is created in the `basicRegistry` with the password `admin`.

For more information about how to modify these logins, see "Configuring WebSphere Application Server Liberty profile" on page 266.

Apache Tomcat

- The login `demo` is created with the password `demo`.
- The login `guest` is created with the password `guest`.
- The login `appcenteradmin` is created with the password `admin`.

For more information about how to modify these logins, see "Configuring Apache Tomcat" on page 267.

Manually installing Application Center

A reconfiguration is necessary for the Worklight Server to use a database or schema that is different from the one that was specified during its installation. This reconfiguration depends on the type of database and on the kind of application server.

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 "In-place upgrade or rolling upgrade to Worklight Server V6.2.0" on page 366) 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 239.
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.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Set **Database type** to **DB2**.
   e. Set **Provider type** to **DB2 Using IBM JCC Driver**.
   f. Set **Implementation Type** to **Connection pool data source**.
   g. Set **Name** to **DB2 Using IBM JCC Driver**.
   h. Click **Next**.
   i. 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}**.
   j. Do not set **Native library path**.
   k. Click **Next**.
   l. Click **Finish**.
   m. The JDBC provider is created.
   n. Click **Save**.
4. Create a data source for the IBM Application Center database:
   a. Click **Resources** > **JDBC** > **Data sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New** to create a data source.
   d. Set the **Data source name** to **Application Center Database**.
   e. Set **JNDI Name** to **jdbc/AppCenterDS**.
   f. Click **Next**.
   g. Enter properties for the data source, for example:
      - **Driver type**: 4
      - **Database Name**: APPCNTR
      - **Server name**: `localhost`
      - **Port number**: 50000 (default)

      Leave **Use this data source in (CMP)** selected.
   h. Click **Next**.

   i. Create JAAS-J2C authentication data, specifying the DB2 user name and password as its properties. If necessary, go back to the data source creation wizard, by repeating steps 4a on page 246 to 4h on page 246.

   j. Select the authentication alias that you created in the **Component-managed authentication alias** combination box (not in the **Container-managed authentication alias** combination box).

   k. Click **Next** and **Finish**.

   l. Click **Save**.

   m. In **Resources** > **JDBC** > **Data sources**, select the new data source.

   n. Click **WebSphere Application Server data source properties**.

   o. Select the **Non-transactional data source** check box.

   p. Click **OK**.

   q. Click **Save**.

   r. Click **Custom properties** for the data source, 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"
   - "Configuring WebSphere Application Server for Derby manually for Application Center" on page 249
   - "Configuring Apache Tomcat for Derby manually for Application Center" on page 251

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. Select the appropriate scope from the **Scope** combination box.

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 Worklight database.

   a. In the WebSphere Application Server console, click **Resources** > **JDBC** >
      Data sources.

   b. Select the appropriate scope from the **Scope** combination box.

   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 248.

   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 240.
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 Foundation 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:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Create a **JDBC provider** named **MySQL**.
   e. Set **Database type** to **User defined**.
   f. Set **Scope** to **Cell**.
   g. Set **Implementation class** to **com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource**.
   h. 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}`.
   i. Save your changes.
4. Create a data source for the IBM Application Center database:
   a. Click **Resources** > **JDBC** > **Data sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New** to create a data source.
   d. Type any name (for example, `Application Center Database`).

e. Set **JNDI Name** to `jdbc/AppCenterDS`.

f. Use the existing **JDBC Provider MySQL**, defined in the previous step.

g. Set Scope to **New**.

h. On the **Configuration** tab, select **Non-transactional data source**.

i. Click **Next** a number of times, leaving all other settings as defaults.

j. 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 240.
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 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 system privilege: CREATE SESSION
      Add system privilege: CREATE SEQUENCE
      Add system privilege: CREATE TABLE
      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 CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE 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 tables for the Application Center 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.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Complete the JDBC Provider fields as indicated in the following table:

Table 66. JDBC Provider field values

Field	Value
Database type	**Oracle**
Provider type	**Oracle JDBC Driver**
Implementation type	**Connection pool data source**
Name	**Oracle JDBC Driver**

   e. Click Next.
   f. 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}**
   g. Click **Next**.

      The JDBC provider is created.
4. Create a data source for the Worklight database:
   a. Click **Resources** > **JDBC** > **Data sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Set **Data source name** to **Oracle JDBC Driver DataSource**.
   e. Set **JNDI name** to jdbc/AppCenterDS.
   f. Click **Next**.
   g. Click **Select an existing JDBC provider** and select **Oracle JDBC driver** from
      the list.
   h. Click **Next**.

i. 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).

j. Click **Next** twice.

k. Click **Resources** > **JDBC** > **Data sources** > **Oracle JDBC Driver DataSource** > **Custom properties**.

l. Set **oracleLogPackageName** to **oracle.jdbc.driver**.

m. Set **user = APPCENTER**.

n. Set **password =** *APPCENTER_password*.

o. Click **OK** and save the changes.

p. In **Resources** > **JDBC** > **Data sources**, select the new data source.

q. Click **WebSphere Application Server data source properties**.

r. Select the **Non-transactional data source** check box.

s. Click **OK**.

t. 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 243, 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="product_install_dir/ApplicationCenter/tools/andr
   <!-- 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>
     <classloader delegation="parentLast">
       <commonLibrary>
         <fileset dir="${wlp.install.dir}/lib"
                  includes="com.ibm.ws.crypto.passwordutil_*.jar"/>
       </commonLibrary>
   ```

```
        </classloader>
      </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 266.

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
   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 Enabling security.

3. Create the Application Center JDBC data source and provider. See the
   instructions in the appropriate subsection in "Manually installing Application
   Center" on page 243.

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 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 binary files of the aapt program, from the Android SDK platform-tools package.

   a. Determine a suitable directory for the aapt binary files in the WebSphere Application Server installation directory.

      • For a standalone server, you can use a directory such as *WAS_INSTALL_DIR*/optionalLibraries/IBM/Worklight/android-sdk.

      • For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/Worklight/android-sdk.

      • For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/Worklight/android-sdk.

- For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/Worklight/android-sdk.
- For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/Worklight/android-sdk.

b. Copy the *product_install_dir*/ApplicationCenter/tools/android-sdk directory to the directory that you determined in 9a on page 261.

c. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

d. Configure the environment entry (JNDI property) android.aapt.dir, and set as its value the directory that you determined in 9a on page 261, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.

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 265.

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 243.

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 247
- "Configuring Apache Tomcat for Derby manually for Application Center" on page 251
- "Configuring Apache Tomcat for MySQL manually for Application Center" on page 254
- "Configuring Apache Tomcat for Oracle manually for Application Center" on page 258

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 267.

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.
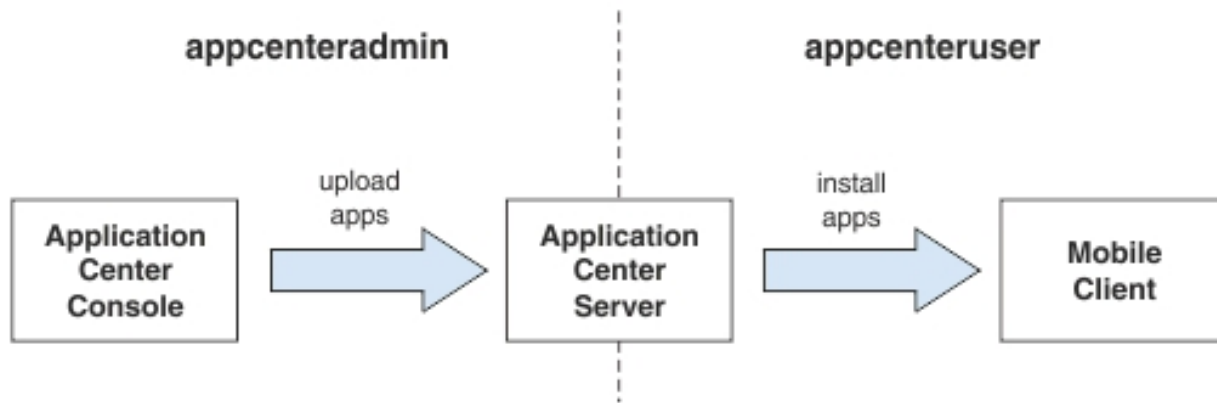
Figure 9. JEE security roles of the Application Center and the components that they influence

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 270 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 1299 for how to build the Application Center mobile client.

Related concepts:

"Managing users with LDAP" on page 270
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		? _
Enterprise Applications		
Use this page to manage installed applications. A single application can be deployed onto multiple servers.		
⊞ Preferences		
Start	Stop	Install	Uninstall	Update	Rollout Update	Remove File	Export	Export DDL	Export File		
🗎 🗎 🗎 🗎		
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		
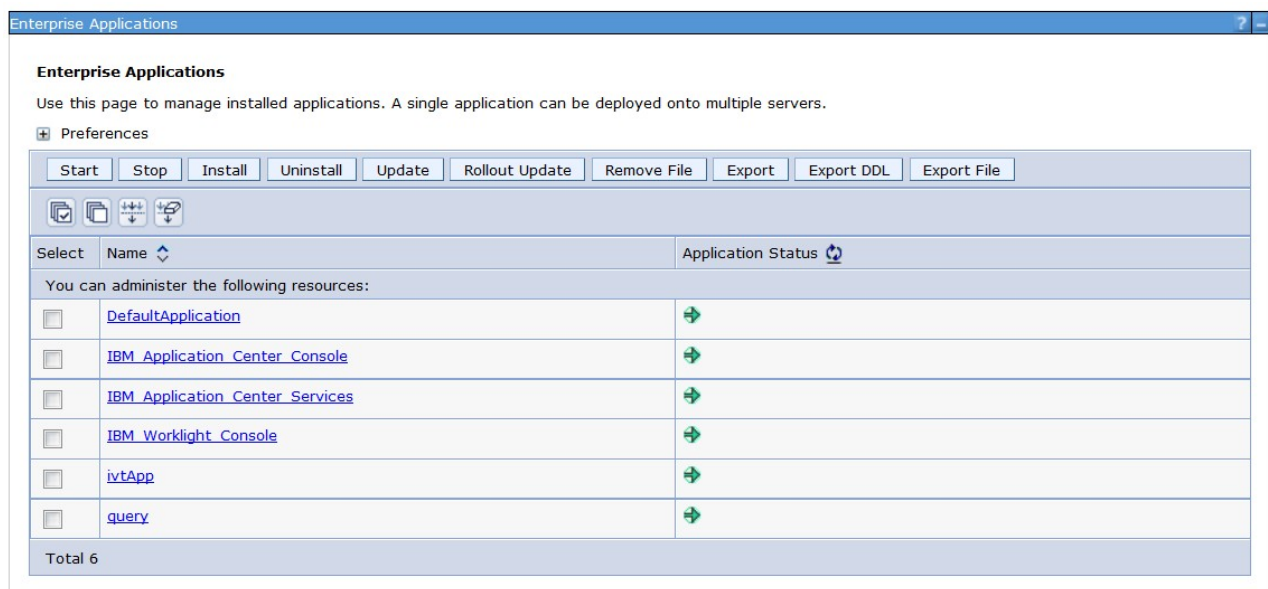
Figure 10. Mapping the Application Center roles

   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.

| Map Users... | Map Groups... | Map Special Subjects ▾ |

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

Figure 11. 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 to avoid exceptions from 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. 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 truststore 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 truststore before you can use application links to Google play.

The root certificate of Apple iTunes must be imported into the WebSphere truststore 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" on page 269.

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

The purpose is to configure the IBM JDK SSL factories to be compatible with Liberty profile. 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.

Exception when WebSphere Application Server SSL socket factories are used

If you use the IBM JDK of WebSphere Application Server, this exception could occur because this JDK uses SSL socket factories that are not supported by the Liberty profile. In this case, follow the requirements documented in Troubleshooting tips.

```
java.net.SocketException: java.lang.ClassNotFoundException: Cannot find the specified class com.ib
    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
    at com.ibm.ws.jmx.connector.client.rest.internal.RESTMBeanServerConnection.<init>(RESTMBeanS
    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)
```

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 on page 271
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.security.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 297 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 12. Environment entries and their values (LDAP and WebSphere Application Server V7)

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

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

About this task

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

You must configure LDAP based on the federated repository configuration. The stand-alone LDAP registry is not supported.

Several different repositories, LDAP and non-LDAP, can be configured in the federated repository.

Procedure

1. Configure the attribute mapping. For users, the Application Center refers to these VMM attributes:
   - **uid**: represents the user login name.
   - **sn**: represents the full name of the user.

   For groups, the Application Center refers only to the VMM attribute **cn**.

If VMM attributes are not identical to LDAP attributes, you must map the
VMM attributes to the corresponding LDAP attributes.

In 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.

     Note: See "List of JNDI properties for the Application Center" on page 297
     for a complete list of properties that you can set

     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 297 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 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**.

Figure 13. 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**.

Figure 14. 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.
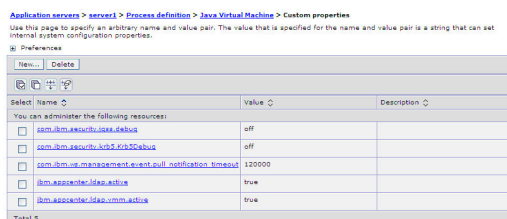
Figure 15. ACL management for Application Center with LDAP on WebSphere Application Server V8

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 67. 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.
`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.
`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 67. 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.security.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` See Using the stand-alone tool to clear the LDAP cache for details.

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 297 for a complete list of LAPD properties that you can 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=inetOrgPerson
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName.filter" value='"(&amp;(cn=%v)(objectclass=i
<jndiEntry jndiName="ibm.appcenter.ldap.group.filter" value='"(&amp;(cn=%v)(|(objectclass=groupOfName
```

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="fa
```

Where:

`JNDI_property_name` is the name of the property you are adding.

`property_value` is the value of the property you are adding.

Table 68. 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.
`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.
`ibm.appcenter.ldap.security.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".

Property	Description
`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` 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.

See "List of JNDI properties for the Application Center" on page 297 for a complete list of LAPD properties that you can set.

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

```
<Environment name="ibm.appcenter.ldap.active" value="true" type="java.lang.String" override="false
<Environment name="ibm.appcenter.ldap.connectionURL" value="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" overrid
<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=inet(
<Environment name="ibm.appcenter.ldap.group.filter" value="(&amp;(cn=%v)(|(objectclass=groupOfNames)
```

Defining the endpoint of the application resources

When you add a mobile application from the Application Center console, the server-side component creates Uniform Resource Identifiers (URI) for the application resources (package and icons). The mobile client uses these URI to manage the applications on your device.

Purpose

To manage the applications on your device, the Application Center console must be able to locate the Application Center REST services and to generate the required number of URI that enable the mobile client to find the Application Center REST services.

By default, the URI protocol, host name, and port are the same as those defined in the web application server used to access the Application Center console; the context root of the Application Center REST services is applicationcenter. When the context root of the Application Center REST services is changed or when the internal URI of the web application server is different from the external URI that can be used by the mobile client, the externally accessible endpoint (protocol, host name, and port) of the application resources must be defined by configuring the web application server. (Reasons for separating internal and external URI could be, for example, a firewall or a secured reverse proxy that uses HTTP redirection.)

The following figure shows a configuration with a secured reverse proxy that hides the internal address (192.168...). The mobile client must use the external address (**appcntr.net**).

Figure 16. Configuration with secured reverse proxy

Table 69. 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 297 for a complete list of endpoint 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 297.

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 70. 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.

For a complete list of LAPD properties that you can set, see "List of JNDI properties for the Application Center" on page 297.

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/applicationc
<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"/>
```

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.

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 71. 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.

For a complete list of JNDI properties that you can set, see "List of JNDI properties for the Application Center" on page 297.

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/applicationcenter"
```

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.

In the <context> section of the Application Center services application:

```
<Environment name="ibm.appcenter.services.endpoint" value="https://appcntr.net:443/applicationcent
<Environment name="ibm.appcenter.proxy.protocol" value="https" type="java.lang.String" override="f
<Environment name="ibm.appcenter.proxy.host" value="appcntr.net" type="java.lang.String" override=
<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, **unless** you intend to install applications on iOS devices. For iOS applications, you must configure the Application Center server with 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, iOS, and Windows Phone 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. On Android, iOS, and Windows Phone operating systems, you can install a self-signed CA certificate on the device to enable the device to handle system calls with respect to this self-signed certificate. This practice is not appropriate for Application Center in a production environment, but it may be suitable during the testing period. For details, see "Configuring SSL by using untrusted certificates" on page 210.

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 keystore.

About this task

This procedure indicates how to request an SSL certificate and import it and the chain certificate into your keystore.

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 keystore.

   The request is sent to the certificate authority.

2. When you receive the SSL certificate, import it and the corresponding chain certificate into your keystore 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 keystore.
   - *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 keystore, 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 keystore for your web server. You can use the Java **keytool** command to create a keystore.

   ```
   keytool -genkey -alias tomcat -keyalg RSA -keystore /path/to/keystore.jks
   ```

2. Import the SSL certificate and the corresponding chain certificate into your keystore 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 keystore.

   ```
   <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 72. 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`.

Property	Description
`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 1311.
`ibm.appcenter.apns.p12.certificate.location`	The path to the file of the development certificate that enables Application Center to send push notifications about updates of iOS applications. 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 1311.
`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 1311.
`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 1310.
`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 1353.
`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 270.

Table 72. 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` 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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 270.
`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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.

Table 72. 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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 287.
`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 287.
`ibm.appcenter.ldap.security.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`. If set, security authentication is used when you connect to LDAP without VMM. See "Configuring LDAP ACL management (Liberty profile)" on page 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.

Table 72. 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 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`ibm.appcenter.ldap.user.loginName`	The login name attribute when you use LDAP without VMM. See "Configuring LDAP ACL management (Liberty profile)" on page 281 and "Configuring LDAP ACL management (Apache Tomcat)" on page 287.
`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 277 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 274.
`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 277 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 274.
`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 277 and "Configuring LDAP ACL management with VMM (WebSphere Application Server V7)" on page 274.
`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.

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 290.
`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 290.
`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 290.
`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 290.

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.

Managing the DB2 transaction log size

When you upload an application that is at least 40 MB with IBM Worklight Application Center console, you might receive a `transaction log full` error.

About this task

The following system output is an example of the `transaction log full` error code.

```
DB2 SQL Error: SQLCODE=-964, SQLSTATE=57011
```

The content of each application is stored in the IBM Worklight Application Center database.

The active log files are defined in number by the **LOGPRIMARY** and **LOGSECOND** database configuration parameters, and in size by the **LOGFILSIZ** database configuration parameter. A single transaction cannot use more log space than **LOGFILSZ** * (**LOGPRIMARY** + **LOGSECOND**) * 4096 KB.

The **DB2 GET DATABASE CONFIGURATION** command includes information about the log file size, and the number of primary and secondary log files.

Depending on the largest size of the IBM Worklight application that is deployed, you might need to increase the DB2 log space.

Procedure

Using the **DB2 update db cfg** command, increase the **LOGSECOND** parameter. Space is not allocated when the database is activated. Instead, the space is allocated only as needed.

Typical topologies of a Worklight instance

A Worklight instance uses a particular topology that is typical for organizations with an established extranet infrastructure.

The following figure depicts this topology.
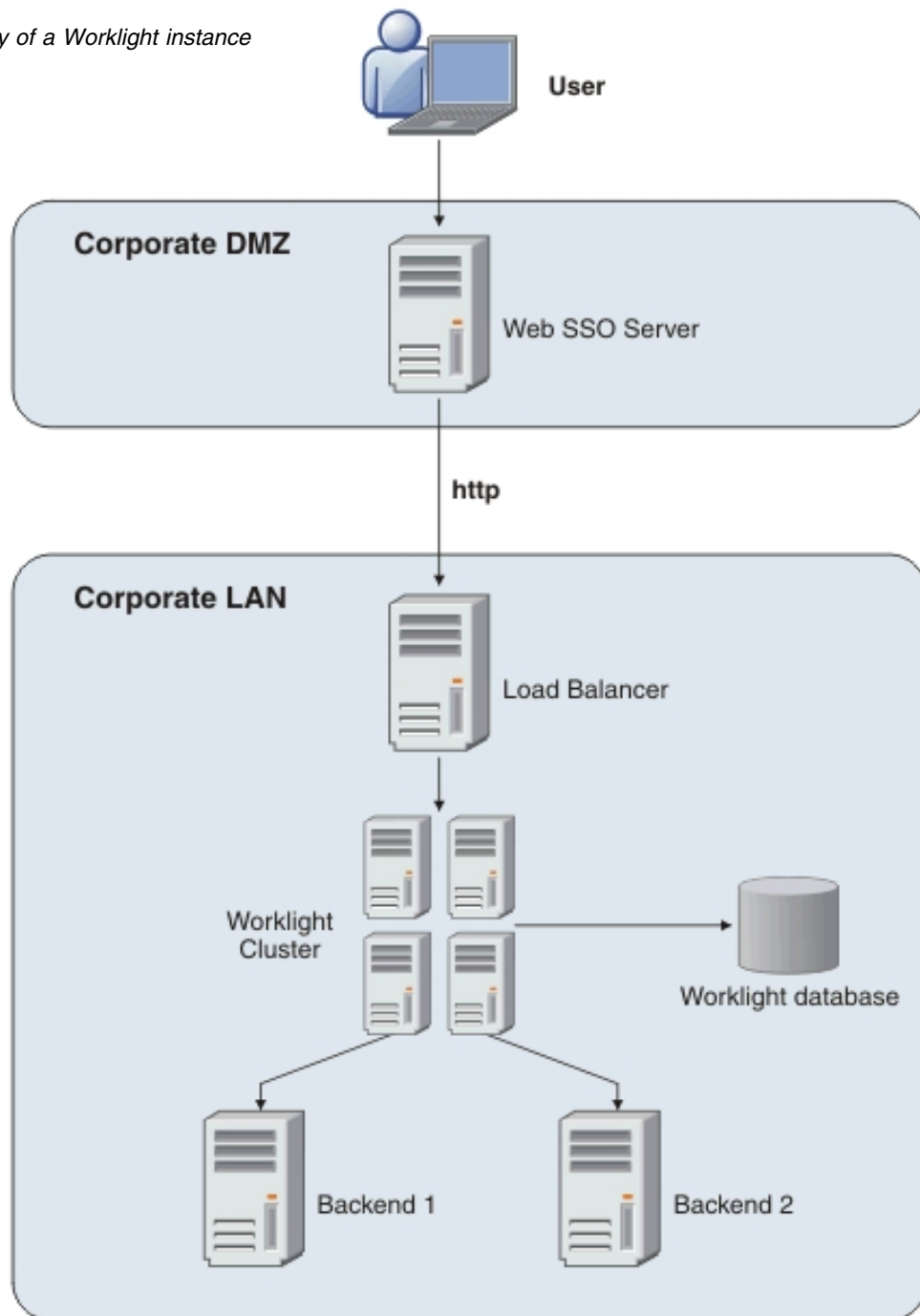
Such a topology is based on the following principles:

- 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 Foundation in an IBM WebSphere Application Server Network Deployment V8.5 cluster environment

You can set up a 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 Foundation in the topology shown in Figure 18:
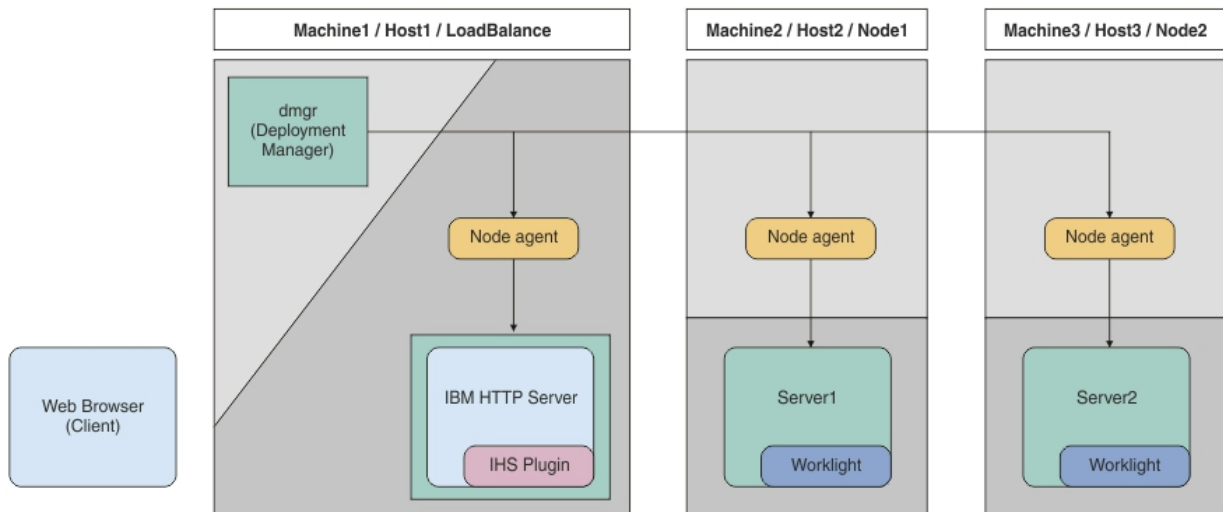
Figure 18. Worklight cluster topology with IBM WebSphere Application Server Network Deployment

This procedure uses the hardware listed in Table 73 and the software listed in Table 74 on page 306.

Table 73. 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 74. Software

Name	Description
IBM Installation Manager 1.8	Install IBM WebSphere Application Server Network Deployment, IBM HTTP Server , IBM Web Server Plug-ins for WebSphere Application Server, and IBM Worklight Foundation.
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 Foundation V6.2.0	IBM Worklight Foundation runtime. You need to get access to the installation repository before you start.
IBM DB2 V9.7 or later	DB2 Database. Your DB2 server must be available before you start the IBM Worklight Foundation installation.
Ant 1.8.3	Configure IBM Worklight Foundation 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.

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.

Note: Node names might be different from the profile names you specify because WebSphere Application Server automatically generates a display name for a new node.

3. Create a cluster and add Worklight nodes as members.

   a. From the WebSphere Application Server administrative console, click **Servers** > **Clusters** > **WebSphere application server clusters**, and then click **New** to create a new cluster.
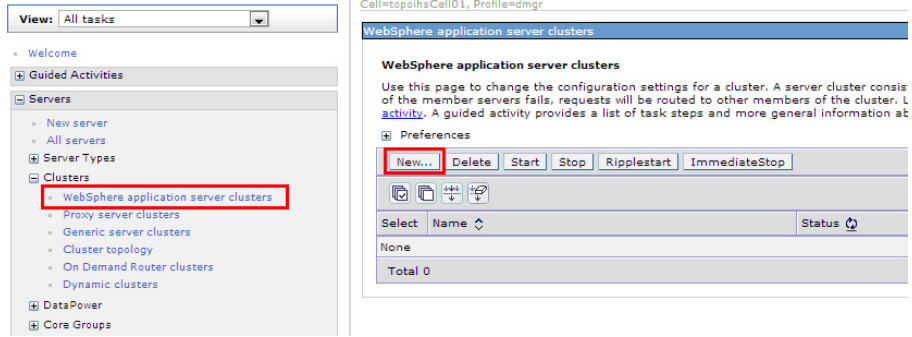
b. For each Worklight node, add a member to the cluster: in the fields provided, enter the required information, and then click **Add Member**.
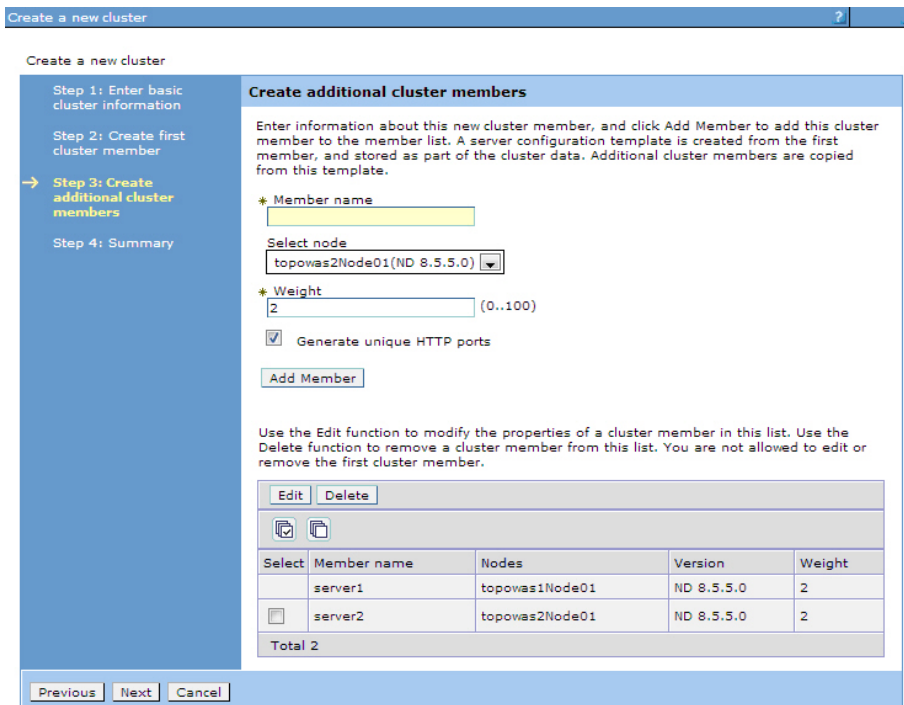
c. From the WebSphere Application Server administrative console, click **Servers** > **Server Types** > **WebSphere application servers** to check that the cluster member servers are listed.
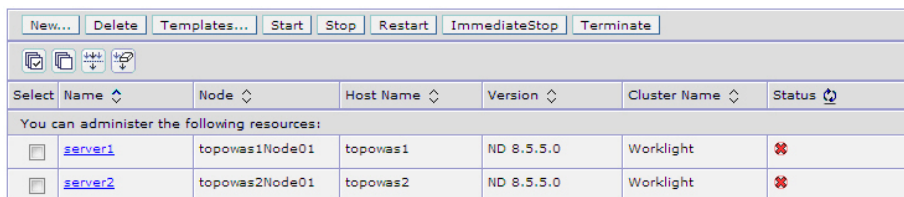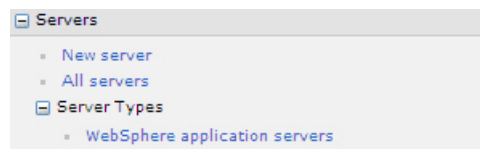
d. If the status column indicates that nodes are not synchronized, click **System Administration** > **Nodes**, and then click **Synchronize** to

synchronize your nodes to the deployment manager.
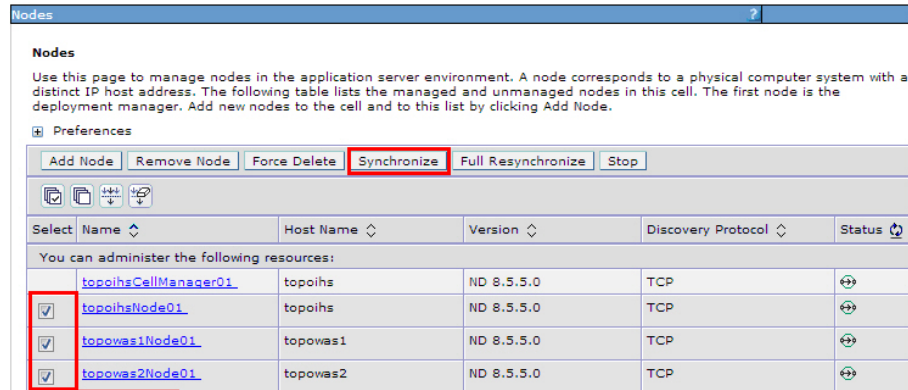
4. Install Worklight Server on Host1. Ensure that the WebSphere Application Server Network Deployment cluster is created without errors before you begin the installation. For installation instructions, see "Installing Worklight Server" on page 70.

5. Configure the databases. For instructions, see "Creating and configuring the databases with Ant tasks" on page 1114.

6. In Worklight Studio, create a Worklight project and build a Worklight project WAR file. See "Artifacts produced during development cycle" on page 412.

7. Configure IBM Worklight Foundation 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 1141. 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/WorklightAdminDS, jdbc/WorklightDS and jdbc/WorklightReportsDS exist. If Application Center is installed, check that the data source jdbc/AppCenterDS exists.
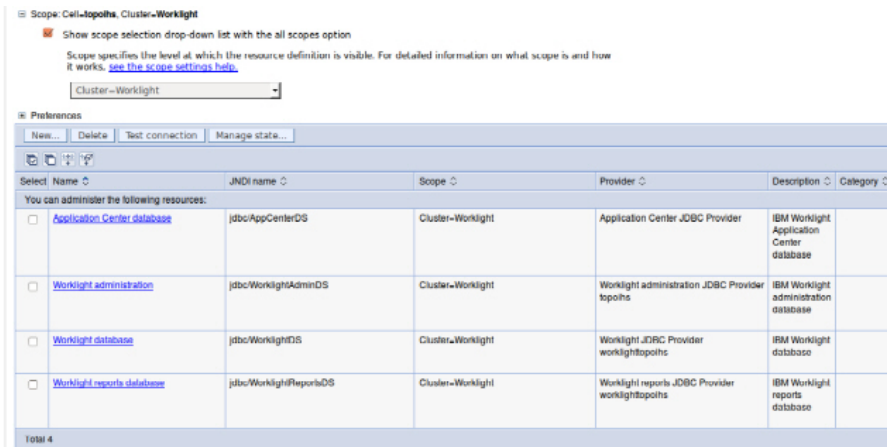
   c. Select the two data sources and click **Test connection** to verify the DB2 database connection. Confirmations similar to the ones in the following messages indicate a successful connection.

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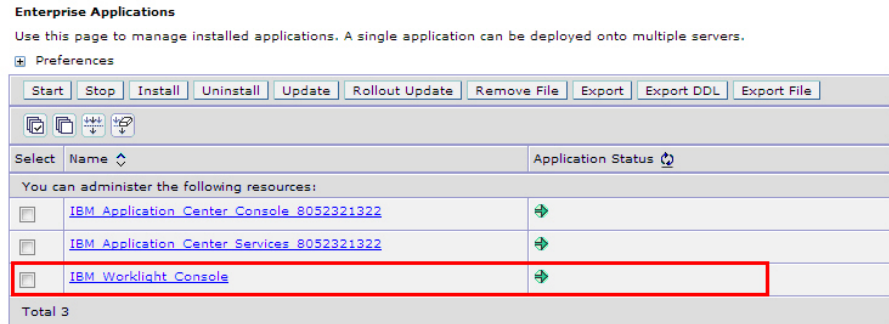
e. Now that you have deployed IBM Worklight Foundation on the two node servers, you can access the Worklight Console on each host by browsing to the associated URLs:

- `http://Host2:9080/worklightconsole`
- `http://Host3:9080/worklightconsole`

Check that both Worklight Console are running.

9. Configure the IBM HTTP Server.

a. From the WebSphere Application Server administrative console, go to **Servers** > **Server Types** > **Web servers**, and then click **New** to create a new IBM HTTP server.
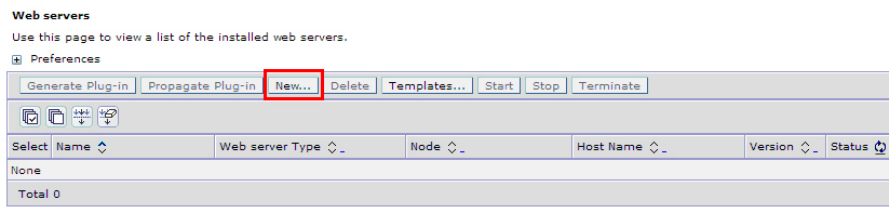
b. Select the "ihs" node you previously created on Host1, then from the **Type** list, select **IBM HTTP Server**, and then click **Next**.
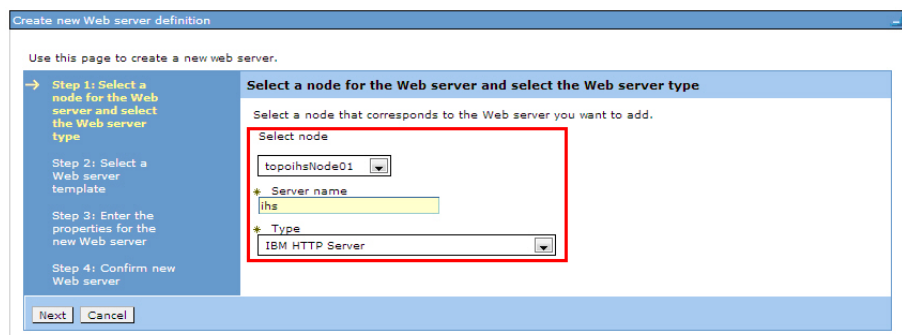
c. Enter the IBM HTTP Server home and Web Server Plugins home you previously selected on Host1, and then click **Next** and save your configuration.
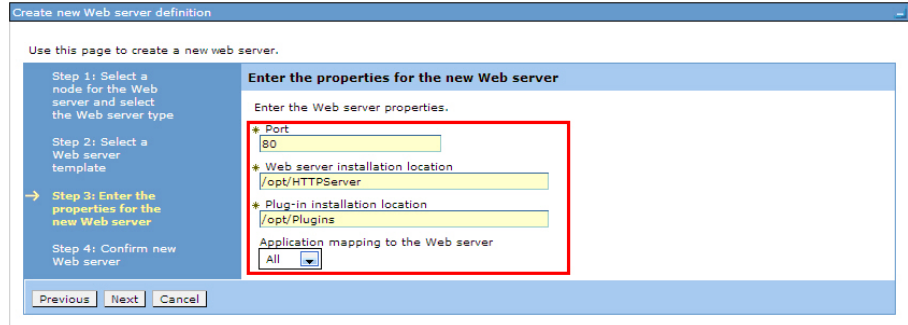
d. In the administrative console, on the Web servers page, click **Generate Plug-in** to generate the plug-in configuration file.
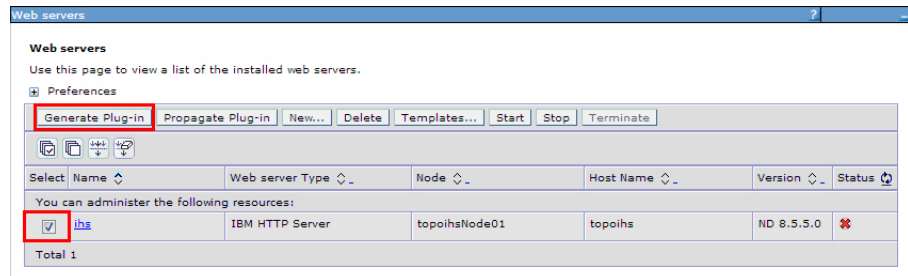
A confirmation message is displayed.
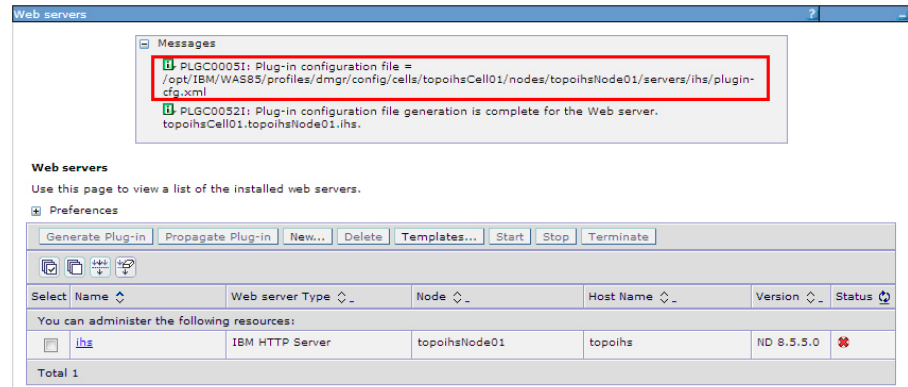
e. Make a note of the `plugin-cfg.xml` location displayed in the confirmation message.

f. In the administrative console, on the Web servers page, click **ihs**, and then in the **Configuration file name** field, click **Edit**.
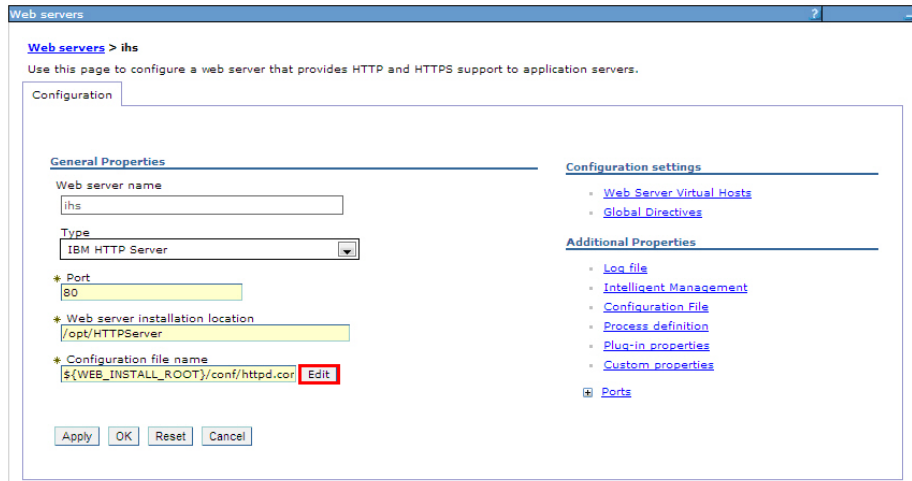
g. In the editor, add a `was_ap22_module` and a `WebSpherePluginConfig` configuration to your `http.conf` file by adding the following text:

On Windows:

```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.dll
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

On Linux:

```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.so
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

h. In the administrative console, on the Web servers page for the "ihs" server, click **Plug-in properties**.
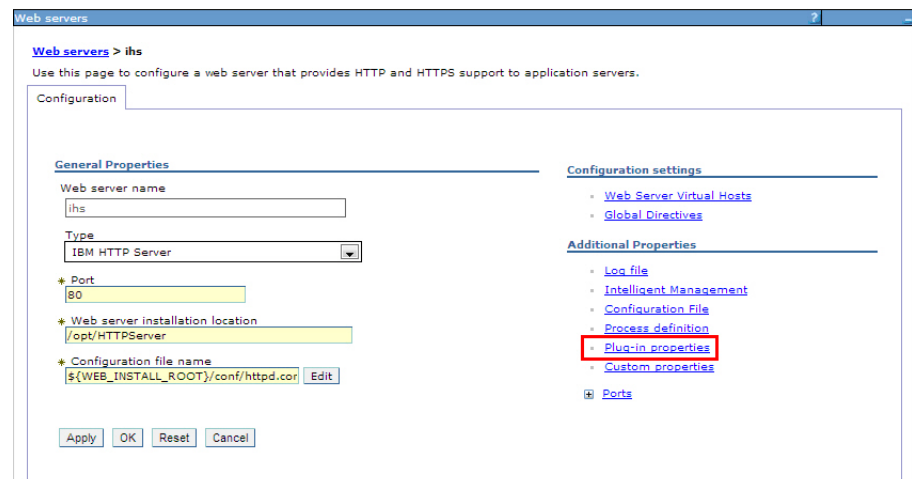
i. In the **Plug-in Configuration file name** field, click **View**.
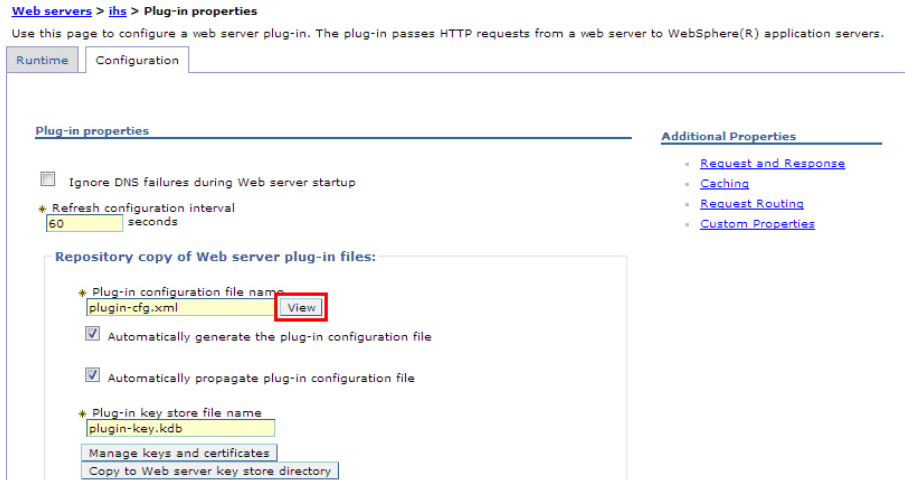
j. Search for the cluster node and Worklight URI in the `plugin-cfg.xml` file. For example:

```
<ServerCluster CloneSeparatorChange="false"
  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="/application
</UriGroup>
```

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

k. Optional: On the Plug-in properties page for the "ihs" server, click **Request Routing** if you want to set a load-balancing policy.

l. Optional: On the Plug-in properties page for the "ihs" server, click **Caching** if you want to configure caching.

10. Start the IBM HTTP server and verify that the server is running.

a. In the WebSphere Application Server administrative console, go to **Servers** > **Server Types** > **Web servers**.

b. Select the IBM HTTP server you created (in this example, named "ihs"), and then click **Start**.
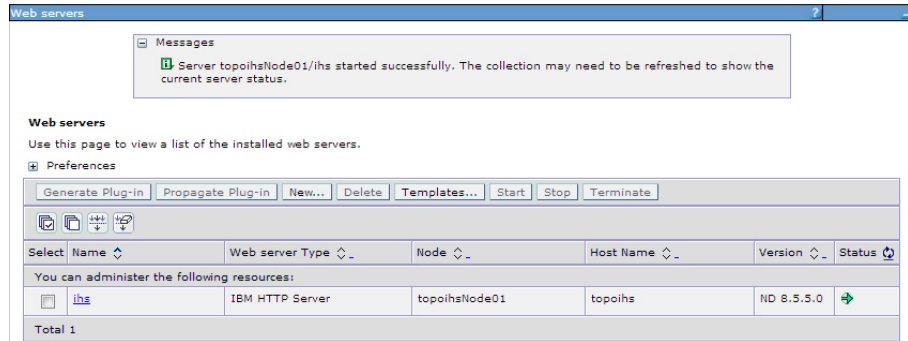
c. If the server fails to start, check the log file. To find the location of the log file:

1) In the administrative console, on the Web servers page for the "ihs" server, click **Log file**.

2) On the log file page, click the **Configuration** tab.

3) The location of the log file is displayed in the **Error log file name** field.
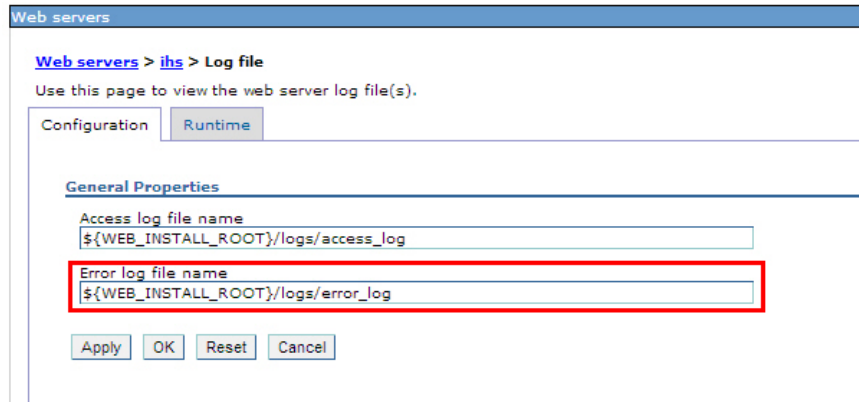
d. To verify that the IBM HTTP server is running, enter the URL for the Worklight Console in a web browser. For example: `http://Host1:80/worklightconsole`.

Results

IBM Worklight Foundation is now installed on an IBM WebSphere Application Server Network Deployment cluster, and is ready for use.

Setting up an IBM HTTP Server in an IBM WebSphere Application Server Liberty profile farm

You can set up a Worklight cluster environment with Liberty profile.

Before you begin

Install a server farm for Liberty. See "Installing a server farm" on page 164. If a server farm is not configured, the Worklight Server installation does not work properly, and changes made by using the Worklight Console or Administration Service are not replicated to all the servers of the farm, resulting in inconsistent behavior for client devices that connect to the Worklight Server.

About this task

You can set up IBM Worklight Foundation in the topology similar to the one shown in Figure 19.
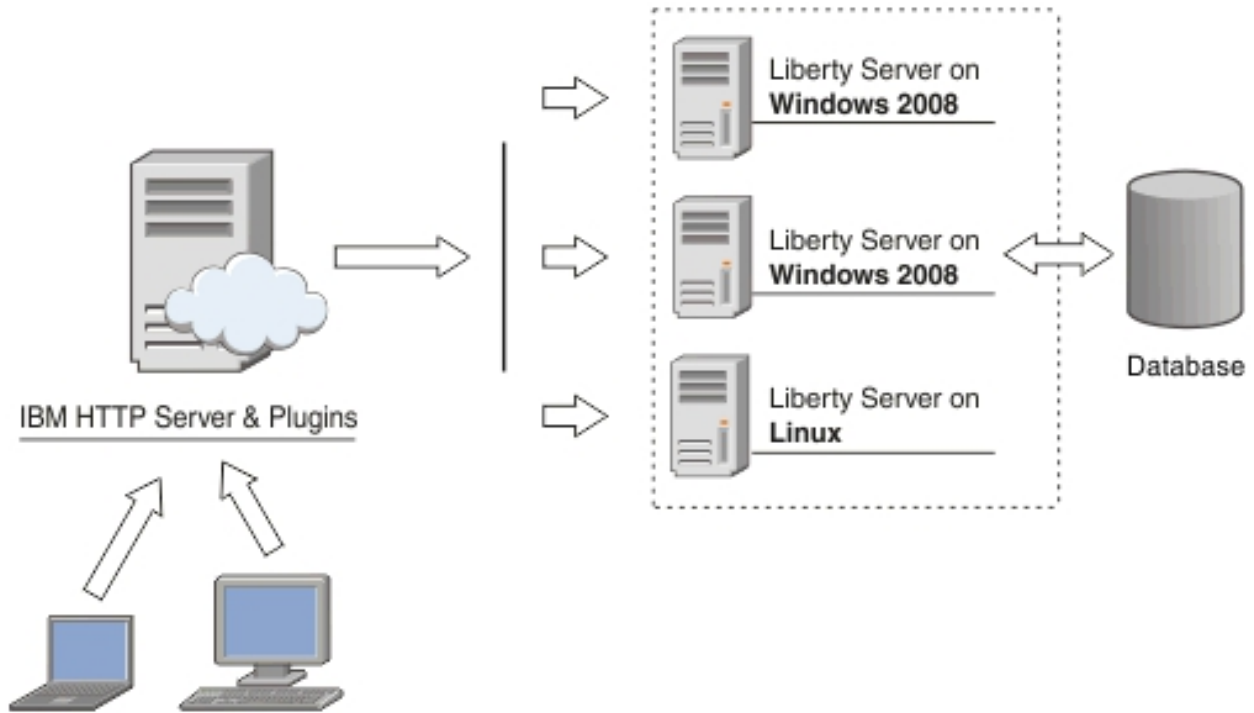
Figure 19. Worklight cluster topology with Liberty profile

This procedure uses the hardware listed in Table 75 and the software listed in Table 76.

Table 75. 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 76. Software

Name	Description
IBM Installation Manager	Install IBM HTTP Server , IBM Liberty profile, and IBM Worklight Foundation.
IBM HTTP Server	You need to get access to the installation repository before you start the procedure. IBM HTTP Server is also included in the WebSphere Application Server installation repository.

Table 76. Software (continued)

Name	Description
Web Server Plug-ins	You need to get access to the installation repository before you start the procedure. IBM HTTP Server Plugin is also included in the WebSphere Application Server installation repository.
IBM Liberty profile	You need to get access to the installation repository before you start the procedure. IBM Liberty profile is also included in the WebSphere Application Server installation repository.
IBM Worklight Foundation	You need to get access to the installation repository before you start the procedure.
IBM DB2	Your DB2 server must be available before you start the IBM Worklight Foundation installation.

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. 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 Console are running.

3. 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 323.

4. Ensure that the IBM HTTP Server can be accessed at the following URL in a web browser:

   `http://<hostname>:<port>`

5. For each Liberty server, generate a web server plug-in configuration file named `plugin-cfg.xml`. The web server plug-in is used to forward HTTP requests from the web server to the application server.

   a. Start the server that hosts your applications and ensure that the localConnector-1.0 feature and other required features are included in the server configuration. Use the pluginConfiguration element in the server configuration file to specify the **webserverPort** and **webserverSecurePort** attributes for requests that are forwarded from the web server. By default, the value of **webserverPort** is 80 and the value of **webserverSecurePort** is

443. Assign the value * to the host attribute to ensure that applications on the Liberty server can be accessed from a remote browser. Here is an example of a `server.xml` server configuration file:

```
<server description="new server">
  <featureManager>
    <feature>localConnector-1.0</feature>
    <feature>jsp-2.2</feature>
  </featureManager>
  <httpEndpoint id="defaultHttpEndpoint" host="*" httpPort="9080">
    <tcpOptions soReuseAddr="true" />
  </httpEndpoint>
  <pluginConfiguration webserverPort="80" webserverSecurePort="443"/>
</server>
```

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

- jConsole:

  1) Using the same JDK as the server, run the jConsole Java utility from a command prompt. For example, run the following command:

     `C:\java\bin\jconsole`

  2) In the jConsole window, click **Local Process**, click the server process in the list of local processes, and then click **Connect**.
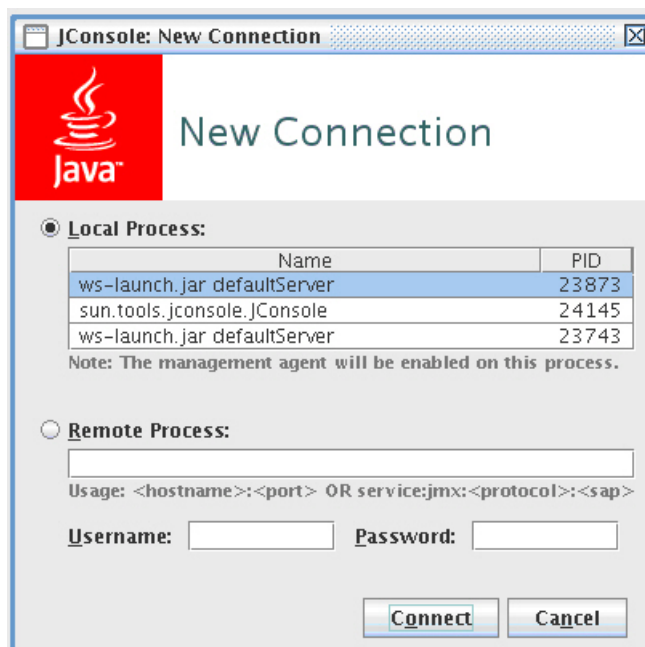
  3) In the Java Monitoring & Management Console, click the **MBeans** tab.
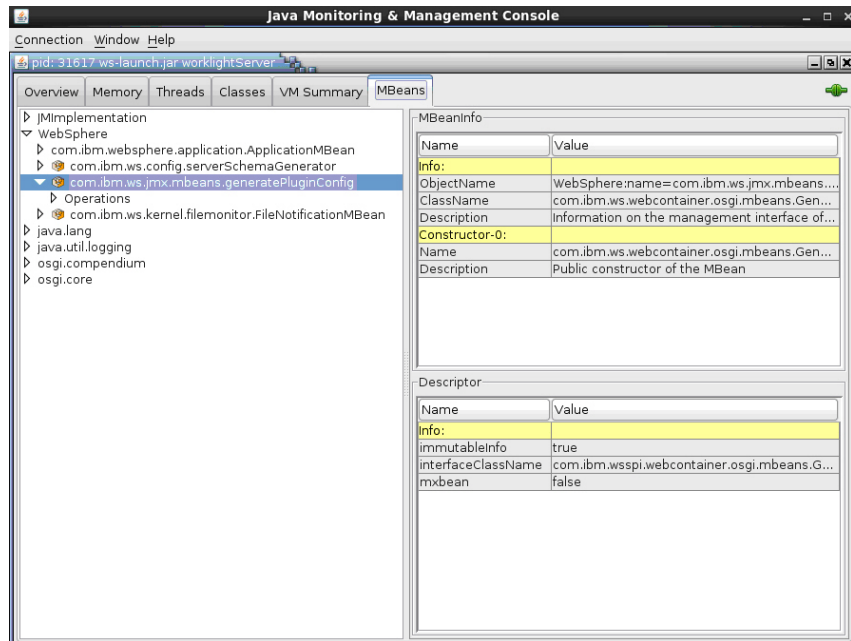
4) Select and invoke the defaultPluginConfig generation MBean operation to generate the `plugin-cfg.xml` file.
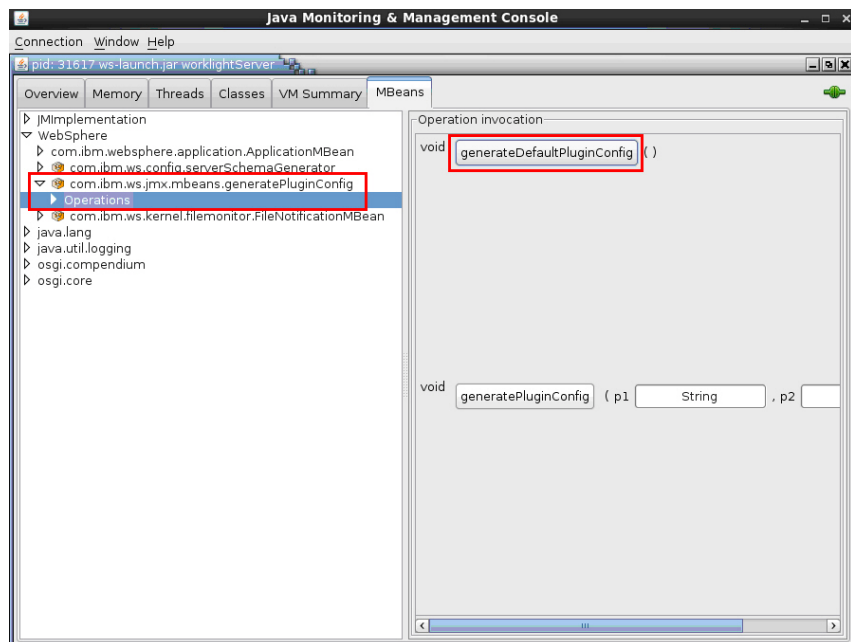
You can find the generated file in the `\wlp\usr\servers\` *<server_name>* directory. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config ASDisableNagle="false"
  AcceptAllContent="false"
  AppServerPortPreference="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-we
              </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"
   ```

6. 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 Generating a merged plug-in configuration for Liberty profile servers using the job manager.
7. 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. Although you can generate **CloneID** values automatically, in production environments, you must specify particular strings in the Liberty Profile `server.xml` file. See Configuring session persistence for the Liberty profile.

   If you do not specify particular strings, the value of the **CloneID** might change under some circumstances and session affinity would stop working. Automatically generated **CloneID** should not be used in a production environment. In WebSphere Application Server Liberty profile, the **CloneID** is generated when you start a server for the first time; it is regenerated if you start the server with the `--clean` option.

   In production use, manually assigning a clone ID ensures that the **CloneID** is stable and that request affinity is correctly maintained. The **CloneID** must be unique for each server and can be 8 to 9 alphanumeric characters in length.

   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 Foundation 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 Foundation and Application Center, and IBM HTTP Server (IHS) for load-balancing Worklight Server 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 a Worklight instance" on page 303.
- IBM Worklight Foundation Application Center is set up on an IBM WebSphere Application Server ND cluster. For more information, see "Installing and configuring the Application Center" on page 238.

- 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 Foundation in the topology similar to the one shown in Figure 20.
.
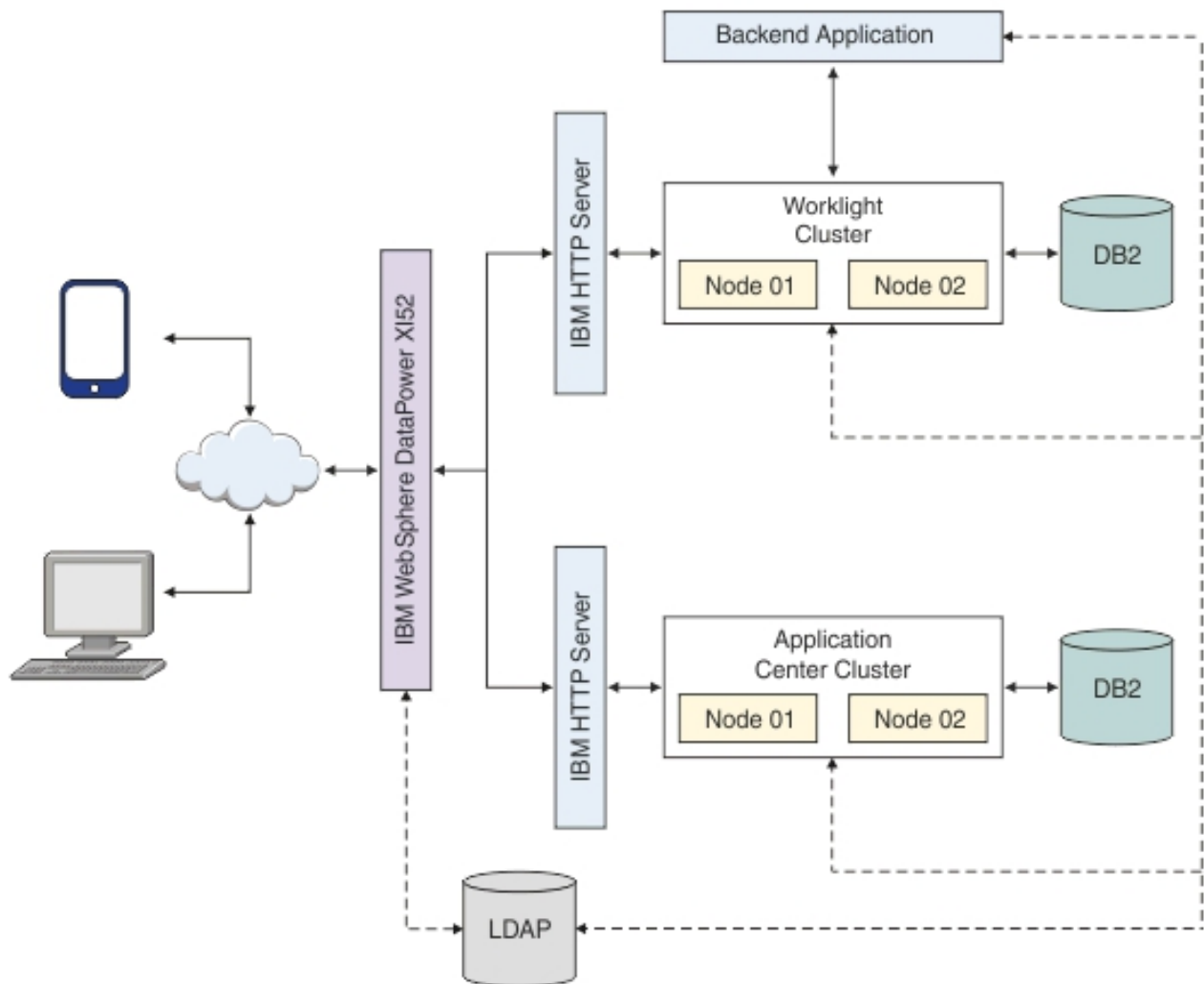
Figure 20. IBM Worklight Foundation integration with an IBM WebSphere Application Server 8.5 Cluster or a Liberty profile Server Farm

DataPower XI52 acts as the gateway for all IBM Worklight Foundation 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 Foundation and Application center.

The following sequence of events takes place when a mobile application makes a request (see Figure 21 on page 327):

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 Foundation 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 Foundation.
6. IBM Worklight Foundation forwards the request to DataPower, and DataPower forwards it to the requesting mobile application.
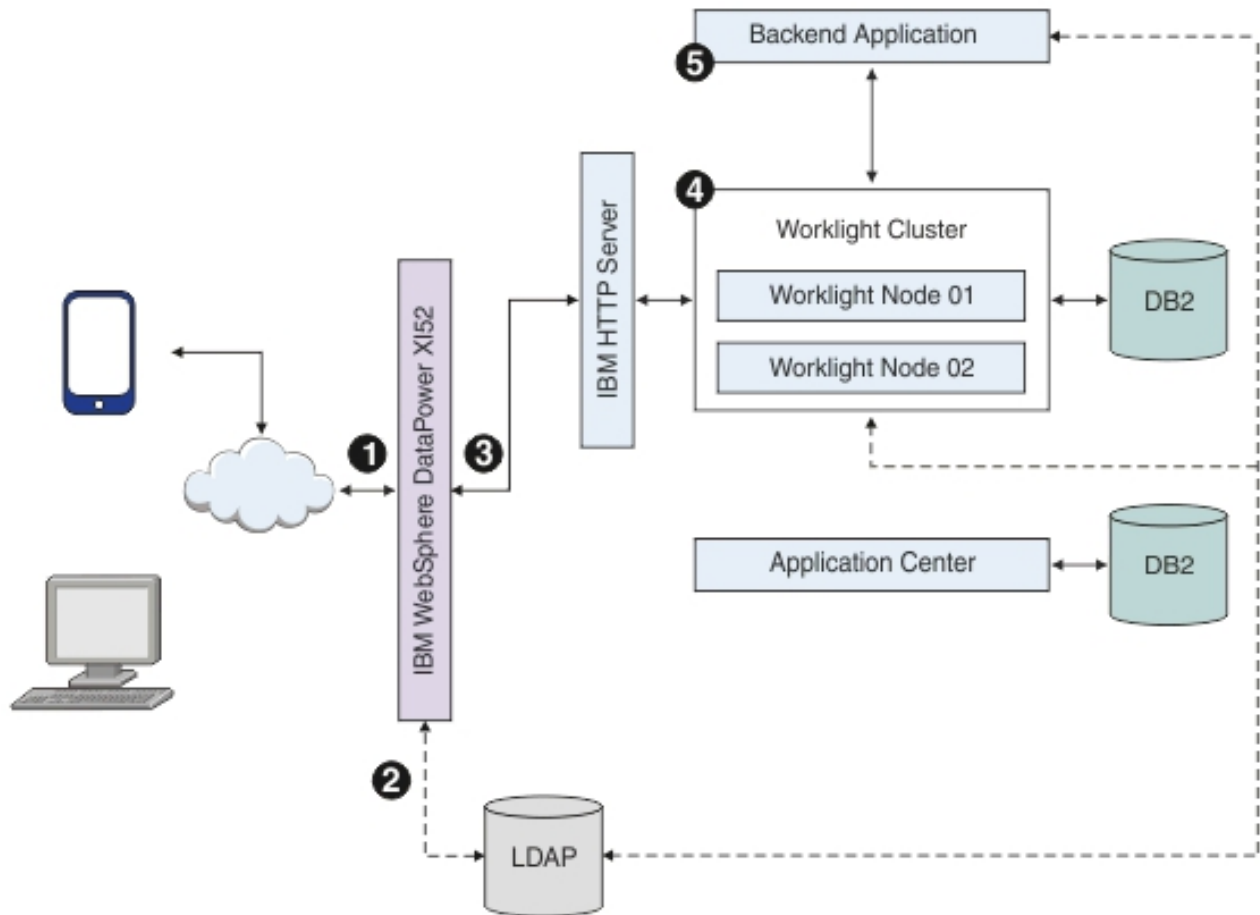
Figure 21. Mobile application request-response flow

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 22 on page 328).
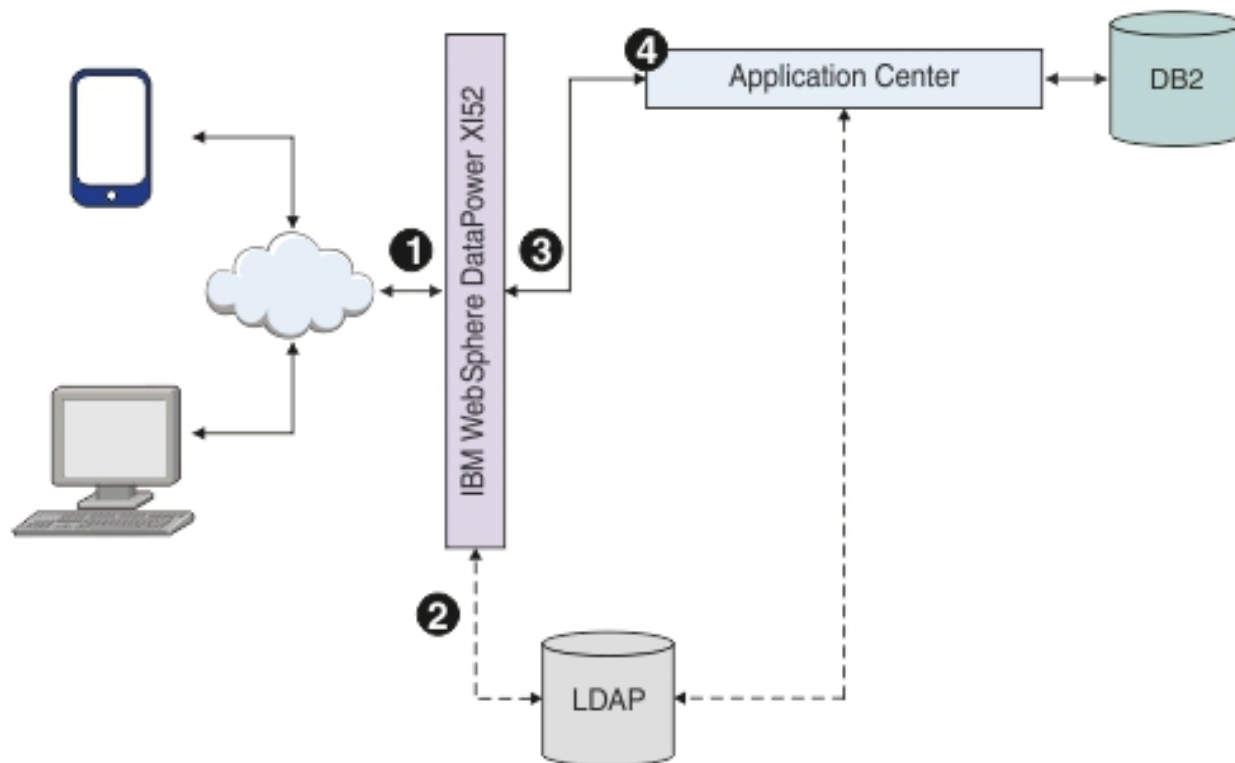
Figure 22. Application Center request-response flow

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 Foundation 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 Foundation WAR file.

Option 2

    Securing WebSphere Application Server using application security but
not securing the IBM Worklight Foundation 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 1212 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. Assume also that there is an HTTP adapter defined on
the server. Assume that the adapter is called SecureAdapter, and that it contains
a procedure called 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);
}
```

Since V6.2.0, Worklight Server is composed of one or more runtime environments, an administration console and administration services, an enterprise application store, and an operational analytics feature. Worklight Server components run as web applications on an application server. For more information about Worklight Server components, see "Introduction to the Worklight Server components" on page 73.

The roles associated with the Worklight Console and Administration Services components are different from the role defined in the WASLTPAModule login module (see "WASLTPAModule login module" on page 902). The Worklight Console and Administration services roles should be mapped to the IT administrator users who are responsible for running administration tasks on the mobile application such as application deployment, management, version enforcement, and management of push notifications.

The roles defined in the WASLTPAModule login module are part of the Worklight runtime environments. These roles should be mapped to the users or user groups that have been cleared to access the Worklight applications. Worklight Console and Administration Services must be set up and configured before you proceed to deploy the Worklight runtime services See the following instructions depending on your application server, to map the administration user roles for Worklight Console and Administration Services:

- "Configuring WebSphere Application Server full profile for Worklight Server administration" on page 151
- "Configuring WebSphere Application Server Liberty profile for Worklight Server administration" on page 152

Once the runtime deployment is completed, you need to map the users against the roles defined in the WASLTPAModule login module or web.xml. 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 Worklight roles.

Figure 23. 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 V6.1.0 and later, the roles can be added as part of the WASLTPAModule login module. See "WASLTPAModule login module" on page 902.

3. Configure Application Center.

   a. Complete the following configuration tasks depending on the server being used:
      - "Configuring WebSphere Application Server full profile" on page 265
      - "Configuring WebSphere Application Server Liberty profile" on page 266

   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 270
      - "LDAP with WebSphere Application Server V8.x" on page 276
      - "LDAP with Liberty profile" on page 280

   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 291
- "Configuring the endpoint of the application resources (Liberty profile)" on page 292

4. Configure DataPower. DataPower XI52 acts as the gateway for all IBM Worklight Foundation 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 24. 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 77:

Table 77. 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 78

   2) `worklight-ssl-policy_verifyLTPA`: see Table 79 on page 334

   3) `worklight-ssl-policy_allowSSLLoginPage`: see Table 80 on page 334

   4) `worklight-ssl-policy_worklightSSLLogin`: see Table 81 on page 335

Table 78. 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 79. Properties of `worklight-ssl-policy_verifyLTPA`

Property	Value
Direction	Client to Server.
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 336. • Input: INPUT • Output: auth
Result	Not applicable.

Table 80. Properties of `worklight-ssl-policy_allowSSLLoginPage`

Property	Value
Direction	Client to Server.
Match	• Type = URL • Pattern = `/(Login|Error)Page\.htm(l)?(\?originalUrl=.*)?`

Table 80. Properties of `worklight-ssl-policy_allowSSLLoginPage` *(continued)*

Property	Value

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

Result	Not applicable.

Table 81. Properties of `worklight-ssl-policy_worklightSSLLogin`

Property	Value
Direction	Client to Server.
Match	• Boolean Or Combinations: On • Type = URL • Pattern: /worklightconsole/* • Type = URL • Pattern: /wladmin/* • 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"

Table 81. Properties of `worklight-ssl-policy_worklightSSLLogin` (continued)

Property	Value
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.
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 Foundation 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 324, Table 79 on page 334), and when

you define rule worklight-ssl-policy_worklightSSLLogin (see "Integrating IBM
WebSphere DataPower with a cluster of Worklight Servers" on page 324, Table 81
on page 335).

```xml
<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="worklightconsole" select="'worklightconsole'"/>
    <xsl:variable name="wladmin" select="'wladmin'"/>
    <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-Contro
    <xsl:variable name="accessControlRequestMethod" select="dp:http-request-header('Access-Control

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

Troubleshooting Worklight Server

You can troubleshoot to locate the server and databases on Windows 8, Windows
7, and Windows XP, or to find the cause of installation or database creation failure.

Troubleshooting to 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 1163.

Troubleshooting failure to create the DB2 database

An incompatible database connection mode might result in failure to create the DB2 database.

About this task

If the following message is displayed when you attempt to create a DB2 database, proceed as follows:

```
"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**, and then **Next** to check whether the issue is solved.
2. If the problem persists, contact your database administrator to solve the database connection issue that is documented on the SQL1035N web page.

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

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

IBM0 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

Configuration for IBM Worklight Server for IBM0 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

OK   Filter   Cancel
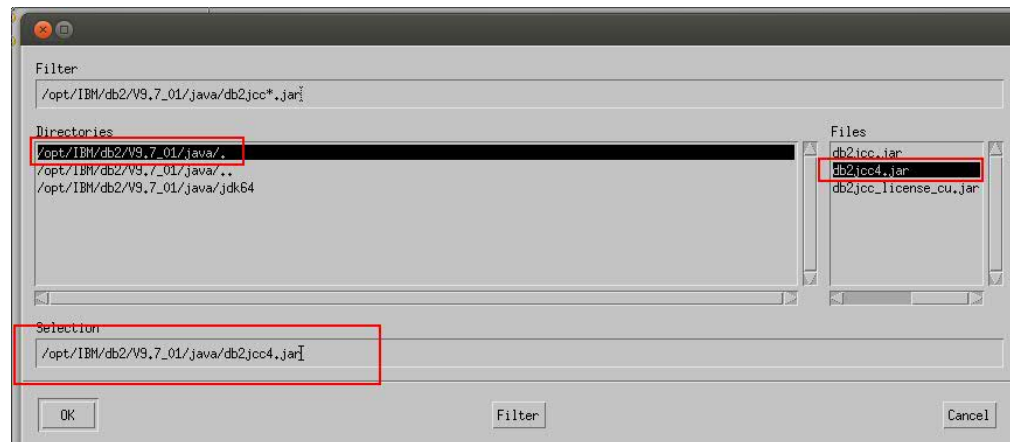
Procedure

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 Foundation 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 UserRegistry
```

When IBM Worklight Foundation 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. This issue might include installations of the Worklight Console and Administration Services.

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 "appcenteradm
  <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 Foundation, 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.

Troubleshooting server farm configuration issues

When you start the Administration Services and the Worklight runtime environments, several exception types can be emitted in the application server logs if the configuration of the server farm is incorrect.

Invalid farm plug-in definition file

Checking farm nodes definition file *<file name>* failed due to the following exception: *<exception>*.

> The validation of the farm plug-in file failed because the XML is not valid, according to the `FarmSchema.xsd` schema definition. The exception gives details about the element that is not valid, according to the schema definition.

> You must edit the farm plug-in file, and modify the XML value that is not valid, based on the exception. Then, restart all the servers of the farm.

Server ID not unique in the farm plug-in file definition file

The `ServerID` *<server id>* is not unique in the plug-in file *<file name>*.

> This is because the value of the **ServerID** attribute in the farm plug-in file is already used. Each node of the farm must have a **ServerID** attribute that is not already used in another node.

> You must edit the farm plug-in file and make sure that all the **ServerID** attributes are unique. Restart all the servers of the farm.

Server ID is not set

`MBeanRegistrationException` "server id JNDI property is not set".

> This is because in the application server configuration, the JNDI property `ibm.worklight.admin.serverid` is not set.

> You must configure the property `ibm.worklight.admin.serverid` in the application server. This property must have the same value than the **ServerID** attribute of this node, which is defined in the farm plug-in file. Restart the application server. For more information, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

Administration Services MBean is already registered

The Administration Services MBean *<MBean name>* is already registered on another node of the farm, which means that the JNDI property `ibm.worklight.admin.serverid` has the same value on other nodes.

This is because the Administration Services MBean is already registered under the same name on another server of the farm. The value of the JNDI property `ibm.worklight.admin.serverid` is the same than the one defined in another server of the farm.

You must configure the property `ibm.worklight.admin.serverid` in the application server. This property must have the same value than the **ServerID** attribute of this node, which is defined in the farm plug-in file, and must be unique among the servers of the farm. Restart the application server.

Chapter 8. Upgrading to IBM Worklight Foundation V6.2.0

This section contains the procedures for upgrading from IBM Worklight V5.0.6 or later to V6.2.0 and migrating your Worklight applications to work with IBM Worklight Foundation V6.2.0.

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.

Those topics cover how to upgrade your Worklight Studio and Worklight Server software to V6.2.0, and how to migrate your applications for V6.2.0.

Version compatibility

Compatibility among different versions of the IBM Worklight client and server depends on several factors.

The following table describes different situations and the compatibility rules that apply to each.

To understand the compatibility rules, it can be useful to understand the IBM product release conventions. Each full number of a release is composed of the following parts, where each part is replaced by a digit from 0 to 9:

version.release.modification.fixpack

Note: Version numbers cited in the table examples are for illustrative purposes only, and might not correspond to actual releases.

Table 82. Version compatibility rules

Description	Compatibility rule	Examples
Server and client have same version and release. (Modification and fix pack release numbers can be different.)	Server and client with the same version and release are fully compatible. If the *modification* or *fixpack* number differ, they are still compatible if the *version* and *release* are the same.	6.2.0.0 server is compatible with 6.2.0.1 client.
Newer server than client.	Compatible.	6.2.0.0 server is compatible with 6.1.0.0 client
Older server than client.	Not compatible.	6.1.0.0 server is not compatible with 6.2.0.0 client

Table 82. Version compatibility rules (continued)

Description	Compatibility rule	Examples
Server artifacts created with older version of Worklight Studio or the Worklight Command Line Interface than the version of the server.	For complete details of which server, .war file, and artifacts work together, see Table 2. However, the following guidelines apply: For versions prior to 6.1.0, only the same versions of server, .war file, and application (.wlapp file) and adapters can work together. If the initial server version is Worklight Server 6.1 or 6.2, the .war file that was created with the same version of Worklight Studio or Command Line Interface for IBM Worklight Developers can be migrated to a newer server version, but the newer server can accommodate only artifacts (.wlapp and adapter files) that were built from the initial version. If the initial server version is Worklight Server 6.2.0.1 or later, including MobileFirst Server 6.3 or later, the .war file that was created with the same initial version of Studio or the Command Line Interface can be migrated to the newer server version. The migrated .war file can accommodate artifacts that were created with any of the following versions of Studio or the Command Line Interface: • Initial version • Newer server version that .war file is migrated to • Any version prior to the initial version The artifacts will behave as though they are running on the older version of the server. For information about migrating the .war file, see "Migrating a project WAR file for use with a new Worklight Server" on page 1164.	Example: Artifacts that are built with Worklight Studio 6.1.0.0 can run on server 6.2.0.0. However, the artifacts will behave as though they are running on server version 6.1.0.0.
Server artifacts created with newer version of Worklight Studio or the Worklight Command Line Interface than the version of the server.	Not compatible.	Artifacts that are built with Worklight Studio 6.3.0.0 cannot run on server 6.2.0.0.

Table 82. Version compatibility rules (continued)

Description	Compatibility rule	Examples
Direct Update feature	If the version of Worklight Studio or the Command Line Interface for IBM Worklight Developers that was used to build an update package differs from the version of Worklight Studio or the Command Line Interface that was used to build the original application package, then the update will not be applied.	Original application was built with Worklight Studio 6.1.0.0; update was built with Worklight Studio 6.2.0.0. Update will not occur.

The following table shows which .war file and artifact versions can work with each server version. Application behavior remains as with the original version of the application.

Table 83. Server, project and artifact compatibility

Server version	Can work with the following project versions (.war file created with this version of Studio or Command Line Interface)	Can work with the Command Line Int
6.2.0.1	6.2.0.1	5.0.6, 6.0.0, 6.1.0, 6.2
6.2.0.1	6.2.0 migrated to 6.2.0.1	6.2.0
6.2.0.1	6.1.0 migrated to 6.2.0.1	6.1.0
6.2.0	6.2.0	6.2.0
6.2.0	6.1.0 migrated to 6.2.0	6.1.0
6.2.0	6.0.0 migrated to 6.2.0	6.0.0
6.2.0	5.0.6 migrated to 6.2.0	5.0.6
6.1.0	6.1.0	6.1.0
6.1.0	6.0.0 migrated to 6.1.0	6.1.0
6.1.0	5.0.6 migrated to 6.1.0	6.0.0
6.0.0	6.0.0	6.0.0
5.0.6	5.0.6	5.0.6

Separation of lifecycle between Worklight Server and Worklight Studio

There is a separation of the Worklight Studio and the Worklight Server upgrade lifecycles, which provides benefits to both developers and IT staff.

Since V6.1.0, IBM Worklight Foundation allows a separation between the Worklight Server and Worklight Studio lifecycles. This separation means that you can 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, V6.1.0.x, and V6.2.0.x) to an instance of Worklight Server V6.2.0.x.

- For project WAR files that were built with versions earlier than V6.2.0, the apps and adapters that you deploy must be built with the same version of Worklight Studio as the version that you used to build the project WAR file. For example, if you develop a project with Worklight Studio V5.0.6 and deploy it to Worklight Server V6.2.0, the WAR file is deployed as a V5.0.6 project. You can deploy to this project only applications and adapters that are developed with Worklight Studio V5.0.6.

- For project WAR files that were built with versions 6.2.0 and later, and deployed to Worklight Server V6.2.0.1, apps and adapters that were built with any version, 5.0.6.x and above (but not later than the project WAR version itself), can be deployed.

The following are some of the limitations of this lifecycle separation:
- Only application environments that are supported by Worklight Server V6.2.0 can be migrated. Older application environments that are not supported by Worklight Server V6.2.0 (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 V6.1.0 to Worklight Server V6.2.0.
- *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 database schema of a Worklight project 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 V6.2.0.
- *Minor version change* - For example, upgrading from V6.1.0 to V6.2.0.
- *Fix pack upgrades* - For example, upgrading from V6.2.0 to V6.2.0.x.
- *Interim fix* - For example, upgrading from V6.2.0 to an interim fix identified by a build number.

Upgrading to Worklight Studio V6.2.0

How to upgrade your current version of Worklight Studio to the latest version.
- To upgrade to Worklight Studio V6.2.0 from previous versions of Worklight Studio Consumer Edition or Worklight Studio Enterprise Edition, you must perform an Eclipse P2 update operation.
- You cannot directly upgrade to Worklight Studio V6.2.0 from a previous version of Worklight Studio Developer edition. You must first uninstall your current instance of Worklight Studio, and install Worklight Studio V6.2.0. After Worklight Studio V6.2.0 is installed, you can then point to your earlier workspace and work with your existing projects.

Upgrading Worklight Studio in the Consumer or Enterprise Editions to Worklight Studio V6.2.0

How to upgrade Worklight Studio in IBM Worklight Consumer Edition or IBM Worklight Enterprise Edition V6.1.x, or earlier, to Worklight Studio V6.2.0.

About this task

The upgrade to Worklight Studio V6.2.0 is performed as an Eclipse P2 update operation. After Worklight Studio V6.2.0 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 V6.2.0 cannot be installed in versions of Eclipse that are older than Juno or V4.2.2.
   - 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.

   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.
8. 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.

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
```

What to do next

The topic "Migrating Worklight projects to Worklight Studio V6.2.0" contains information about how to migrate and work with your existing projects.

Upgrading Worklight Studio in the Developer Edition to Worklight Studio V6.2.0

How to upgrade from IBM Worklight Developer Edition V6.1.x, or earlier, to Worklight Studio V6.2.0.

About this task

You cannot directly upgrade to Worklight Studio V6.2.0 from an earlier version of IBM Worklight Developer Edition. Instead, you must uninstall IBM Worklight Developer Edition, and install Worklight Studio V6.2.0.

Note: The Eclipse menu **Help** > **Check for Updates** does not find a new version of IBM Worklight Foundation. You must explicitly search for Worklight in Eclipse Marketplace.

Procedure

1. Start your Eclipse IDE workbench and verify your version of Eclipse:
   - Worklight Studio V6.2.0 cannot be installed in versions of Eclipse that are older than Juno or V4.2.2.
   - If you have an older version of Eclipse, update it to Juno or Kepler before continuing this procedure.
2. Uninstall your current Worklight Studio Developer Edition. For more information, see "Uninstalling Worklight Studio" on page 63.
3. Install Worklight Studio V6.2.0 from Eclipse Marketplace by following the procedure Install Worklight Developer Edition From the Eclipse Marketplace on the IBM Mobile development website

What to do next

You can now point to your earlier workspace, and work with your existing projects. The topic "Migrating Worklight projects to Worklight Studio V6.2.0" contains information about how to migrate and work with your existing projects.

Migrating Worklight projects to Worklight Studio V6.2.0

How to migrate your existing projects to Worklight Studio V6.2.0.

Migrating older Worklight projects to Worklight Studio V6.2.0

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 V6.2.0.

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 that are listed in Migrating to a newer version of IBM Worklight of the IBM Worklight V5.0.5 user documentation.
2. Open the project (now migrated to V5.0.5) in Worklight Studio V6.2.0 to complete the migration process.

Note: Access to authenticity-protected resources is denied when you upgrade existing applications to Worklight Studio V6.2.0. When you upgrade Worklight Studio, rebuild the project WAR and `.wlapp`, and deploy those files to the server, the existing clients that use authenticity stop working. To resolve the problem, complete the following steps:

1. Upgrade the project using the new Worklight Studio, as described above.
2. Increment the versions of the upgraded applications.
3. Deploy the new WAR file that was built with the new Studio.
4. Deploy the new applications to the server alongside the applications that were built with the old Worklight.
5. Normally, both applications work as expected. If you want to use the new ones only, block the old ones and refer to the new ones for upgrade.

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 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.

Note: You must build and deploy your project after a migration.

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 618.
- For Android Native API projects, follow the instructions in "Copying files of Native API applications for Android" on page 624.
- For JavaME Native API projects, follow the instructions in "Copying files of Native API applications for Java Platform, Micro Edition (Java ME)" on page 630.

ARC-enablement for upgraded iOS projects

The Worklight upgrade process does not enable ARC (Automatic Reference Counting) by default in existing iOS projects. Although ARC is enabled for the Worklight library, you must manually enable ARC for the upgraded project by following the steps that are described in the Apple Documentation.

iOS: changes in Xcode linker options

Before IBM Worklight Foundation V6.2.0, the iPhone and iPad environments specified the -all_load option in the **Other Linker Flags** of the **Build Settings** to load all symbols from all libraries into the app. As of IBM Worklight Foundation V6.2.0, this -all_load flag is replaced with -force_load *<library>* flags for each library from which all symbols are loaded.

Generally, the linker loads the symbols that are used within an app. However, in some cases, such as when Objective-C categories are used, the linker does not load all the necessary symbols, which results in unresolved symbol errors from the linker.

New V6.2.0 iOS environments and the iOS environments that are migrated from previous versions of IBM Worklight Foundation have the -force_load flag specified for the following libraries:
- libCordova.a
- libWorklightStaticLibProject.a
- libTLFLib.a (added by the Tealeaf SDK optional feature)
- libfipshttp.a (added by the FIPS 140-2 optional feature)

Note: You might need to add a -force_load flag for third-party libraries that you include in your project.

If you use xcodebuild to build your projects, you must specify an extra flag or property to indicate the location of the libraries for which you use -force_load. You can use either the -scheme flag (if the project was previously opened in Xcode) or the SYMROOT property (whether the project was previously opened in Xcode or not):

Note: In the following examples, the values used for -configuration and -sdk are only indicated for illustration purposes. The values that you specify might be different.
1. If Xcode was not previously run on the generated .xcodeproj project, then specify SYMROOT (it can be any directory):

   xcodebuild -project <PathToProject>/TestProjectTestAppIphone.xcodeproj -configuration Release -s
2. If Xcode was previously run on the generated .xcodeproj project, then you can specify the scheme:

   xcodebuild -project <PathToProject>/TestProjectTestAppIphone.xcodeproj -scheme TestProjectTestApp

Targeting new iOS architectures for your existing apps
- For hybrid applications: To target armv7, armv7s and arm64, or only one architecture, you must manually change the architecture selection in the **Architectures** menu of the Xcode **Build Settings**:
  1. In **Valid Architectures**, select armv7 or armv7s or arm64, or all the three architectures.
  2. In **Architectures**, depending on the slice that you want to create, select one, or several architectures that you chose in step 1.

     Note: To compile or to create a slice, you must include the slice name in both **Valid Architectures** and **Architectures** (so for armv7s, you must add

armv7s to both menus). **Valid Architectures** specifies declaratively which slice can be created. In **Architectures**, select only the slice that you want to create.

- For native applications: After you replace the IBM Worklight V6.1.0 files with the V6.2.0 files in your Xcode project, you can change the selections in the Xcode **Valid Architectures** and **Architectures** menus to include any combination of armv7, armv7s, and arm64. You can then compile your project for the selected architectures.

Note: FIPS is not yet supported on 64-bit architecture even though Worklight library does support 64-bit architecture. Therefore, FIPS must not be enabled on 64-bit target platform when XCode Build Setting (Architecture) is also set to 64 bit.

Manually upgrading existing Worklight IOS hybrid applications

IBM Worklight Foundation V6.2.0 supports existing Worklight applications that were created before V6.2.0. These applications can continue to work as is. However, if you want to simplify the usage of native code at the start of your application, you must manually update the application code.

To manually migrate an application to IBM Worklight Foundation V6.2.0, and use native code more easily at the start of your application, follow these steps:

1. Set the main app delegate, `MyAppDelegate` (in *appName*.h), to extend WLAppDelegate instead of WLCordovaAppDelegate.
2. Implement the protocol WLInitWebFrameworkDelegate (for example, set `MyAppDelegate` to implement it), and implement its method wlInitWebFrameworkDidCompleteWithResult.

   Note: The method didFinishWLNativeInit is deprecated and cannot be called. If you had custom code in this method, move it to wlInitWebFrameworkDidCompleteWithResult.

   For example, in *appName*.h, change

   ```
   @interface MyAppDelegate : WLCordovaAppDelegate{}
   ```

   to

   ```
   @interface MyAppDelegate : WLAppDelegate <WLInitWebFrameworkDelegate> {}
   ```

   and implement the method of the protocol:

   ```
   -(void)wlInitWebFrameworkDidCompleteWithResult:(WLWebFrameworkInitResult *)result△
   ```

   Note: CDVMainViewController and WLCordovaAppDelegate are deprecated and are no longer necessary (if some custom applicative code interacts with these classes, it continues to work as is).
3. Call `[[WL sharedInstance] initializeWebFrameworkWithDelegate:someDelegate]` whenever it applies, according to the logic of the application (it is recommended to call it as early as possible in the application lifecycle).
4. When the Worklight initialization process is complete, the framework notifies the delegate by calling the wlInitWebFrameworkDidCompleteWithResult method with a WLWebFrameworkInitResult object. Check the initialization result by using the `statusCode` method of WLWebFrameworkInitResult. The success and failure scenarios are handled according to the logic of the app.
5. If the initialization finishes successfully, you can load the Cordova WebView to run the Worklight JavaScript framework:

a. Call [[WL sharedInstance] *mainHtmlFilePath*] to get the path of the main HTML file of the app.

b. Create the Cordova `ViewController CDVViewController`, and set its `startPage` to the HTML path.

c. Display the Cordova WebView.

Managing the splash screen in existing iOS hybrid applications migrated to V6.2.0

In existing iOS hybrid applications migrated to V6.2.0, the splash screen is still managed internally by a compatibility layer in the same way it was managed in previous versions. In applications before V6.2.0, the showing and hiding of the splash screen was managed internally in native code by the Worklight Hybrid SDK. To be able to use the new splash API, migrate your application as described in "Manually upgrading existing Worklight IOS hybrid applications" on page 351.

Working with Android projects

In IBM Worklight Foundation V6.2.0, developers are encouraged to use the latest Android SDK API level that is supported by the Worklight Studio – Android 4.4 (API Level 19). 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 19. This value specifies that the API Level of the application targets is Android 4.4.

The default Android SDK API level is not changed for existing projects that are opened in Worklight Studio V6.2.0.

Note: (Only for projects that were created with versions older than IBM Worklight V6.1) 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`.

Change from Tealeaf V8 to Tealeaf V9

- Tealeaf Client SDK V9 is delivered under the optional feature **Tealeaf Client SDK** in Worklight Studio V6.2.0. In previous version of IBM Worklight Foundation, the name of this optional feature was Analytics.
- `WL.Analytics` JavaScript API no longer delegates to Tealeaf TLT JavaScript API. `WL.Analytics` JavaScript API now delegates to Worklight internal SDK and achieves the same results as the previous behavior.
- For Worklight V6.1 projects in which the Analytics optional feature is already enabled and that you upgrade to Worklight Studio V6.2.0: the Tealeaf artifacts are replaced, including `uicandroid.jar` and `TLFConfigurableItems.properties` on Android, all Tealeaf `.h`, `.plist` and `libTLFLib.a` files on iOS, and `tealeaf.min.js`.

  Note: The three Tealeaf import statements are commented out in the `main.m` file of iPhone and iPad environments. This is necessary to avoid compile errors that are related to unresolved import statements, as the Tealeaf header files were removed from the Worklight SDK. The header files are available as part of the

Tealeaf Client SDK optional feature. Applications that use the Tealeaf Client SDK optional feature must add the appropriate header file imports based on the Tealeaf version that is used.

- Applications are not automatically instrumented with Tealeaf Client SDK API calls. Make sure to instrument the application appropriately to capture the required events for Tealeaf CX server replay and analysis.

Impact of migrating to a new version of Worklight Studio for applications already in production

Since IBM® Worklight® V6.2.0, there is a separation between the Worklight Server runtime environment and the Worklight apps or adapters. This generally means the following:

- When you deploy a runtime environment that was built using Worklight Studio V6.2.0 into Worklight Server V6.2.0.1, you can deploy applications and adapters of any version, from V5.0.6.

- It is recommended that you increase the application version when you rebuild it in the new Worklight Studio. If not, important features such as the authenticity will fail for the old clients and they will not be able to connect to the Worklight Server.

For Direct Update and app authenticity to work, both the client application and the server-side artifacts (wlapp) must be generated from the same version of Worklight Studio.

In certain 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. If you deploy a new, upgraded runtime environment (meaning one that was build with the new Worklight Studio), it is possible to deploy both versions of applications - the one built with an older version of Worklight Studio and the one that was upgraded and built with a new version of Worklight Studio, with a different application version. You can still serve the older, existing client application along with new ones, or block the old ones and refer to download of the new ones.

There are three such cases:

- 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 IBM 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 and app authenticity are 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 599) 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 1252. 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 1249.

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.

- Applications that use application authenticity. App authenticity will not work properly for clients built with older version of Worklight Studio when the app that is deployed on the server was built with a newer version of Worklight Studio, and both have the same application version. Requests from those clients to access resources that are authenticity-protected will always be denied.

Migrating IBM SmartCloud Analytics Embedded to IBM Worklight Analytics Platform

If you used IBM SmartCloud Analytics Embedded in previous versions of IBM Worklight Foundation, you must now migrate to IBM Worklight Analytics Platform.

About this task

In IBM Worklight Foundation V6.2.0, IBM Worklight Analytics Platform replaces IBM SmartCloud Analytics Embedded. Complete the following steps to migrate to IBM Worklight Analytics Platform. For more information about IBM Worklight Analytics Platform, see "Operational analytics" on page 1397.

Procedure

1. Install the analytics WAR file on your application server, but do not start the server. For detailed information about installing the analytics WAR file, see "Installing the IBM Worklight Analytics Platform" on page 232.

2. Locate the `data` folder on your IBM SmartCloud Analytics Embedded server. If the installation path for IBM SmartCloud Analytics Embedded was not modified, this folder is located in `/opt/IBM/analytics/data`.

3. Copy the `data` folder to the same machine as the machine where the analytics WAR file is hosted.

   Note: The `data` folder then becomes the location where all analytics data is stored, so make sure to place this folder in an appropriate location.

4. Modify the *datapath* JNDI variable on your application server to point to the `data` folder that was copied from the IBM SmartCloud Analytics Embedded server folder in step 3. For example:

   ```
   <jndiEntry jndiName="analytics/datapath" value="/home/system/data"/>
   ```

   Important: Make sure the JNDI property points to a copied version of the data folder. This is to ensure that your data is still backed up in case of data corruption due to a migration failure.

5. Identify the cluster name that was specified when IBM SmartCloud Analytics Embedded was installed. This name will be the name of the folder at the root of the `data` folder.

6. Modify the `clustername` JNDI variable on your application server to match the cluster name that was installed by IBM SmartCloud Analytics Embedded. For example:

   `<jndiEntry jndiName="analytics/clustername" value="WLCLUSTER"/>`

7. Start the Analytics WAR server and review the console. The migration process begins automatically. The data is available to view after the migration process is completed.

Upgrading to Worklight Server V6.2.0 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 later to V6.2.0 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 Worklight applications that run in a production Worklight Server environment. For step-by-step instructions on how to upgrade your production Worklight Server to V6.2.0, see the following topics.

Note: The documentation in the topics that follow assumes the following facts:
- 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 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 postinstallation, verification, and configuration tasks that must be completed before you restart the new Worklight Server and finish migrating your 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.

Start with "Overview of the upgrade to Worklight Server V6.2.0 process," and then read through the steps under each of the major topics that follow.

Overview of the upgrade to Worklight Server V6.2.0 process

An overview of the Worklight Server V6.2.0 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 Application Center and for Worklight Server. This DBMS hosts and run the following databases:
  - The Application Center database (if Application Center is installed on that server).
  - The Worklight administration database.
  - One or more Worklight runtime databases. Each runtime environment requires one database for Worklight and an optional database for Worklight Reports.
- One or more application servers. These application servers host and run the following web applications:
  - The Application Center application (if Application Center is installed on that server).
  - The Worklight Console application. One Worklight Console can be used to administer several Worklight runtime environments. It is defined by a WAR file, which is `worklightconsole.war`.
  - The Administration Services application. This application provides the necessary services for the Worklight Console and hosts all the services (REST services) and administration tasks. The Administration Services application is defined by a WAR file, which is `worklightadmin.war`, and is connected to the administration database.
  - One or more Worklight runtime environments. Each Worklight runtime environment:
    - Is defined by a WAR file that is created with the Worklight Studio development tool.
    - 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 Worklight Server programs, usually on the same computer as the application server or deployment manager.

Other items can belong to an IBM Worklight Foundation configuration, for example, an IBM HTTP Server, IBM DataPower, or an LDAP system.

The topics in this section focus on the task of upgrading and configuring the following entities:

- The Worklight Server programs.
- The databases, including the creation of the administration database.
- The Worklight project runtime applications and Application Center applications that are deployed in the application server.

  Note: The upgrade of the Worklight project runtime applications includes the installation and setup of the Worklight Console and administration services applications.

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 Foundation to Worklight Server V6.2.0 (for example, from V6.0.0.x to V6.2.0 or from V6.1.0.x to V6.2.0).

- From Worklight Server V6.2.0 to a fix pack release (for example, from V6.2.0 to V6.2.0.x).
- From Worklight Server V6.2.0 to an interim fix release (for example, from V6.2.0 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 Worklight upgrade path.
- Whether your Application Center and Worklight Server (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 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0 to V6.2.0.x	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes/No	Yes/No	Yes/No	Yes/No	Yes/No	Running/ Stopped/ Uninstalled/ Upgraded	Running/ Stopped/ Uninstalled/ Upgraded

Preparation for upgrades to Worklight Server

Before you begin the actual upgrade to Worklight Server V6.2.0, you must complete several preparation tasks.

Upgrading 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 inIBM Worklight Foundation 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.

Complete the following preparation tasks before you begin upgrading to a new Worklight Server version. You can start any time before the upgrade, but you must complete these tasks before you move to the next major step, "Starting the Worklight Server V6.2.0 upgrade process" on page 370.

Gathering information for Worklight Server V6.2.0 upgrades

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 for upgrades to Worklight Server. 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 that you are likely to be asked for and keep it handy during the upgrade process.

In addition, it is often necessary to pre-plan certain aspects of the upgrade and clear them with your application server administrator and your database administrator. For example, you must know the correct user name. You must also 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0 to V6.2.0.x	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	Running	Running

Procedure

Go through the following checklist.
- Verify that your operating system, application server, and Database Management System (DBMS) meet the system requirements for Worklight Server V6.2.0 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. IBM Installation Manager needs access to these drivers to upgrade the Application Center database. Ant scripts also need access to these drivers to create the administration database and upgrade the Worklight runtime databases.
- Gather the credentials to the Worklight Server administration, the Worklight runtime environments, the Worklight reports, and Application Center databases. If you do not know the correct schemas, user names, and passwords, ask your database administrator for assistance.

  Note: The Worklight Server administration database does not exist for IBM Worklight 6.1 or earlier.
- Stop and restart the application server and verify its configuration. If you do not know the correct schemas, user names, and passwords, ask your database administrator for assistance.
- If the URL to the Application Center or the Worklight Server applications or their console changes, identify all the systems that you must update. If you

upgrade from V6.1.0 to V6.2.0, the URLs to the Application Center and the Worklight runtime environment do not change, but a new URL is introduced for the Worklight Console.

Planning installation of the Worklight Administration Services and Worklight Console

You must plan the steps that you perform later to upgrade the Administration Services and the Worklight Console. These components are introduced in V6.2.0, and if you upgrade from an earlier version, you must install them first.

Before you begin

Worklight Server V6.2.0 introduces a new architecture for the unified console based on several core elements that are described in "Introduction to the Worklight Server components" on page 73.

If you upgrade from IBM Worklight V6.1.0 or earlier, you must install the following new components as part of the upgrade process: the Administration Services, and the Worklight Console.

The present topic lists the items that you must plan before you perform that upgrade process. The actual installation procedure for the Administration Services, and optionally the Worklight Console, is at "Installation or upgrade of Worklight Server Administration Services" on page 382.

About this task

The following table lists the upgrade paths for which this step is mandatory.

Table 84. Upgrade paths

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	No	No	Running	Running

If you upgrade from V6.1.0 or earlier, install the Administration Services as part of the upgrade process.

To minimize downtime or issues while you follow the upgrade procedure, start with reviewing the installation procedure and configuration options:

1. Review the topic .
2. Define your upgrade strategy if multiple Worklight runtimes are installed (project WAR files).
3. Prepare the configuration of the application.
4. Set up the Worklight Administration database.
5. Review the configuration of the application server.

The following procedure emphasizes important items that you must prepare before running an upgrade. You must also review the installation instructions at "Installing and configuring Worklight Server administration" on page 104.

The following steps are for planning only, and you do not have to start the installation of the Worklight Server administration at this stage. The actual installation is described later in the upgrade procedure, at "Installation or upgrade of Worklight Server Administration Services" on page 382.

Procedure

1. Review the topic to define the application server where you can install the Worklight Server administration components.
2. Define your upgrade strategy if multiple Worklight runtimes are installed (project WAR files).

   Note: You must perform this step only if you have more than one Worklight runtimes (project WAR files) to upgrade. If you have only one Worklight runtime to upgrade, you can skip this test.

   In this case, you can either manage all the runtimes with the same Worklight Administration Services and Console runtime environment or install this environment for each runtime.

   • Manage all the runtimes with the same Worklight Administration Services and Console runtime environment: This is the default setting. Carefully review the context root of each runtime. The context root is used to identify a runtime in the administration database. After the Worklight administration data is migrated to the administration database, you can no longer change the context root of a Worklight Server runtime. For more information, see "Upgrade the Worklight and Worklight Reports databases" on page 383.

   • Install a Worklight Administration Services and Console environment for each runtime: In this case, define the environment IDs as follows:
     – If you install by running an Ant file, add an `environmentID` attribute to the Ant tasks for installation administration: `<installworklightadmin>`, `<updateworklightadmin>`, `<uninstallworklightadmin>`, `<configureapplicationserver>`, `<updateapplicationserver>`, `<unconfigureapplicationserver>`. For more information, see "Ant tasks for installation of Worklight runtime environments" on page 1146 and "Ant tasks for installation of Worklight Console and Administration Services" on page 120.
     – If you install manually, update the `ibm.worklight.admin.environmentid` JNDI property as documented in "List of JNDI properties for Worklight Server administration" on page 158 and "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

3. Prepare the configuration of the application.

   Since IBM Worklight Foundation V6.2.0, the protection of the Administration Services and the Worklight Console is configured by security roles that are managed in the application server. For more information, see "Configuring user authentication for Worklight Server administration" on page 151. To prepare the installation and configuration of the Administration Services and the Worklight Console, you must identify the users who need access to the console, and verify that these users are declared in the application server. This way, you can configure their access to the Administration Services and Worklight Console when the applications are installed.

4. Set up the Worklight Administration database.

A Worklight administration database is necessary for the Administration Services. This database can be created at installation time, if you have an administrator access to the database server. Otherwise, you must contact your database administrator so that the database is created in advance, and you must provide your database administrator with the information listed at "Optional creation of the administration database" on page 104.

5. Review the configuration of the application server.

   For IBM Worklight Foundation V6.2.0, you must configure your application server to enable Java Management Extensions (JMX) communication between the Administration Services and the Worklight Server runtime. Review the topic "Configuration of the application server" on page 107 to see if there is a need to configure your application server to support JMX for a production environment. For example, in the case of WebSphere Application Server Liberty profile, the Ant tasks that you use to install the Administration Services can configure a default secure JMX connection, which includes the generation of a self-signed SSL certificate with a validity period of 365 days. But this configuration is not intended for production use.

Identify the Worklight WAR file and prepare the Ant deployment script

In this task, you identify the Worklight project 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 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0 to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	Running	Running

Procedure

In the upgrade process, the Worklight runtime environment 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:

1. Find the WAR file that was previously deployed to the application server.
   - If you are upgrading from Worklight Server V6.0.0.x or later, find the JNDI properties that were set for the deployed Worklight project to override the default `worklight.properties` file. If you used an Ant script with the **configureapplicationserver** task to deploy the WAR file, you can find in that script the JNDI properties that were set at installation time. For more information, see "Configuration of Worklight applications on the server" on page 1171.

For upgrading from Worklight Server V6.0.0.x or later, the procedure to deploy the WAR file is described at "Deploying the project WAR file" on page 1105.

- 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 1104.

  When you modify the WAR file, use Worklight Studio V5.0.6.x or the Ant tasks (`worklight-ant.jar`) from an installation of Worklight Studio V5.0.6.x that was used to build the apps previously deployed to the server. The version of Worklight Studio that was used to build the project WAR file must exactly match the version of Worklight Studio that was used to build the apps previously deployed to the server.

  The WAR file is automatically upgraded to V6.2.0 format during the deployment procedure that is described in later steps.

2. Prepare the Ant deployment script that is used to upgrade this WAR file to V6.2.0 and to deploy this WAR file to the application server, with the upgraded Worklight Runtime Library.

   - When you upgrade from Worklight Server V6.0.0.x or later, you can reuse the script that you used for initial deployment. Make a copy of this file and modify it as follows:

     a. In the Ant file, make sure that the reference to the JAR file is `worklight-ant-deployer.jar`, and not `worklight-ant.jar`. Since IBM Worklight V6.1.0, this library is named `worklight-ant-deployer.jar`. For example, in the V6.0.0 script looks contains these lines:

        ```
        <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:

        ```
        <taskdef resource="com/worklight/ant/defaults.properties">
          <classpath>
            <fileset dir="${worklight.server.install.dir}/WorklightServer">
              <include name="worklight-ant-deployer.jar"/>
            </fileset>
          </classpath>
        </taskdef>
        ```

     b. In the Ant file, you must add the following targets, which are specific to IBM Worklight Foundation V6.2.0:
        - **adminstall**
        - **minimal-admupdate**
        - **admuninstall**
        - **admdatabases**
        - **minimal-update**

     c. Add `<adminDatabase>` to the `<configuredatabase kind ="Worklight">` Ant task. This element upgrades the administration data to the new administration database.

        Those targets are required to install the Worklight Console and Administration Services. You can find examples of such targets in "Sample configuration files" on page 1156. If you use the XML extracts of

the sample configuration files, replace the variables ({$...}) by the corresponding variables of your Ant file. For more information about the references of the Ant tasks, see:

– "Ant **configuredatabase** task reference" on page 1115
– "Ant tasks for installation of Worklight Console and Administration Services" on page 120
– "Ant tasks for installation of Worklight runtime environments" on page 1146

- If you upgrade from Worklight Server V5.0.6.x:
  – Install Worklight Server V6.2.0 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.

  Note: Alternatively, you can create an Ant file by using the Server Configuration Tool and export the resulting Ant file to run it on the command line. However, be aware that the Server Configuration Tool does not send separate calls to the individual Ant targets as is required in the upgrade procedure. Therefore, if you create an Ant file with the Server Configuration Tool, carefully review the database settings. If your database settings do not match the settings that were used at installation time, the deployed apps are not visible to the Worklight Server.

3. Verify that the **environmentID** attribute for the Worklight runtime environments matches the **environmentID** attribute that is used to install the Worklight Server administration Ant file.

   If you install the Worklight Server administration components with a different Ant file than the one you used to install the Worklight runtime environment, for example if you install the Worklight Server administration with the Server Configuration Tool, you might have a different **environmentId** for the administration and the runtime. In this case, the Worklight Server administration components would not find the Worklight runtime environments.

   The **environmentID** is an attribute of the following Ant tasks:

   - installworklightadmin, updateworklightadmin, and uninstallworklightadmin Ant tasks, which are documented at "Ant tasks for installation of Worklight Console and Administration Services" on page 120.
   - configureapplicationserver, updateapplicationserver, unconfigureapplicationserver Ant tasks, which are documented at "Ant tasks for installation of Worklight runtime environments" on page 1146.

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 IBM Worklight Foundation:
  - 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 IBM Worklight Foundation:
  - 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 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 V6.2.0	**V6.0.0.x to V6.2.0**	**V6.1.0.x to V6.2.0**	**V6.2.0 to V6.2.0.x (Fix Pack)**	**V6.2.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
Yes	Yes (for both Worklight Server and Application Center)	Yes (for both Worklight Server and 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 V6.2.0, 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.

In IBM Worklight Foundation V6.2.0, no mobile operating system is dropped or deprecated. Since IBM Worklight V6.1.0, some of the Worklight environments such as iGoogle, Facebook, Apple OS X Dashboard, Vista that were supported in IBM Worklight V5.0.6 are no longer supported. If a target mobile device has an application that is installed on it which requires an environment that is no longer supported by a version of Worklight Server anterior to V6.2.0, the application on this device will stop working after an upgrade of Worklight Server to V6.2.0.

Therefore, if you upgrade Worklight Server from V6.0.0.x or earlier to Worklight Server V6.2.0, you must pay particular attention to the following 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 V6.2.0	V6.0.0.x to V6.1.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	No	No	No	Running	Running

Procedure

If your current version ofWorklight Server includes existing applications that target environments that are no longer supported by Worklight Server V6.2.0:

- For old, no-longer-supported environments, your application developers must update the Worklight application to run with an environment supported by Worklight Server V6.2.0 before you can run it.
- For new environments for which support is added after the release of Worklight Server V6.2.0, 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

In-place upgrade or rolling upgrade to Worklight Server V6.2.0

You can upgrade to a new version of the product in one of two ways: as an *in-place upgrade* or as a *rolling upgrade*. An in-place upgrade replaces the previous version while a rolling upgrade does not.

You can replace the previous version by the new one or you can install the new version alongside the previous one.

In-place upgrade
> An upgrade by which the old version of Worklight Server is no longer installed after the new version of Worklight Server is installed.
>
> In-place upgrades require some downtime of the service.

Rolling upgrade
> An upgrade that installs the new version of IBM Worklight Foundation 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.
>
> The procedure for a rolling upgrade to apply a fix pack to IBM Worklight Foundation V6.2.0 is documented in "Rolling upgrade procedure to apply a fix pack to IBM Worklight Foundation V6.2.0" on page 398.

The following table shows possible upgrade paths.

Table 85. Upgrade paths

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	Running	Running

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 what is the impact on 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.

Since 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	No	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 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 Foundation.
2. Install WebSphere Application Server Liberty Core with IBM Installation Manager. The installer for WebSphere Application Server Liberty Core is part of the IBM Worklight Foundation package.
3. Create a server in this new WebSphere Application Server Liberty profile installation. If you are not familiar with the creation of a server for Liberty, see the "Tutorial for a basic installation of Worklight Server" on page 77.
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 361 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 375, follow the instructions for "Upgrading from Worklight Server V5.0.6.x (changing the Liberty server)" on page 378.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	Running	Running

Procedure

Before you use IBM Installation Manager to upgrade your production server, familiarize (or refamiliarize) 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 V1.7.2, 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 "Command-line installation with XML response files (silent installation)" on page 86 and "Working with sample response files for IBM Installation Manager" on page 87 and prepare your response file.

   To prepare your response file from sample response files, create a response file based on the following versions of Worklight Server, and sample files:

Table 86. 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 V6.2.0), you need only:
- The base repository V6.2.0 installation package from Passport Advantage or physical media.

To install a fix pack release (for example, Worklight Server V6.2.0.1), you need:
- The corresponding base repository (such as Worklight Server V6.2.0) installation package from Passport Advantage or physical media. The corresponding base repository for V6.2.0.x fix packs is the V6.2.0 release.
- The appropriate V6.2.0.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 V6.2.0 release as an **Install** choice and the V6.2.0.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

Attention: 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 V6.2.0 upgrade process

In this phase of the upgrade process, you shut down and back up the application server and Worklight databases and perform other pre-installation tasks.

When you finish the tasks that are listed in "Preparation for upgrades to Worklight Server" on page 357, you can begin the actual upgrade process.

Note: After you complete this phase of the upgrade process, your Worklight Server, Application Center, databases, and application server(s) 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 375 attempts to remove and replace many files in theWorklight 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	Running	Running

Procedure

Back up all the application servers (or network deployment nodes) where the Application Center and Worklight Server administration applications are 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 down the application server 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 that is 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, which is in `C:\ProgramData\IBM\Worklight\WAS85liberty-server`.

Otherwise, if the application server is running when IBM Installation Manager starts the upgrade, some upgrade operations might 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes (if installed on multiple servers)	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 V6.2.0. No instance of Application Center can be running while this operation is performed.

After the database is migrated, only migrated Application Center applications must be run, because only migrated applications are able to read and write to the new databases. Otherwise, the Application Center database might be corrupted.

If you installed Application Center only once, this operation is done automatically by IBM Installation Manager.

Back up the Application Center database

Before you run IBM Installation Manager to install Worklight Server V6.2.0, 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 V6.2.0. This operation cannot be undone. If, for any reason, you decide to roll back the upgrade of Worklight Server, you 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 IBM Worklight V5.0.6: APPCNTR
- For IBM DB2 and MySQL if you installed IBM Worklight V6.0.0 or later: APPCNTR
- For Oracle, if you installed IBM 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 382 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 "Preparation for upgrades to Worklight Server" on page 357 and "Starting the Worklight Server V6.2.0 upgrade process" on page 370 sections that preceded this step.

It is also possible to run IBM Installation Manager in silent install mode, using response files that are 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 Foundation. For more information, see "Command-line installation with XML response files (silent installation)" on page 86 and "Working with sample response files for IBM Installation Manager" on page 87.

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 V6.2.0.

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 Foundation 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 Foundation V6.2.0. 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 andWorklight Server are on the same application server	
V5.0.6.x to V6.2.0	**V6.0.0.x to V6.2.0**	**V6.1.0.x to V6.2.0**	**V6.2.0 to V6.2.0.x (Fix Pack)**	**V6.2.0 to interim fix**	**Application Center Status**	**Worklight Server Status**
No	Yes	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 V6.2.0 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes (unless Liberty server was installed by Worklight Server V5.0.6)	No	No	No	No	Stopped (all instances)	Uninstalled

Procedure

1. Start IBM Installation Manager.
2. Click **Install**. The package name for Worklight Server has changed between Worklight Server V5.x and V6.2.0, so the upgrade must be done with the 'Install' process.
3. If you are doing an in-place upgrade (see "In-place upgrade or rolling upgrade to Worklight Server V6.2.0" on page 366), 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 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 the current version of Worklight Server. 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.
     - 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 368 section of the "Packaging change of WebSphere Application Server Liberty profile in IBM Worklight V6.x" on page 367 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes (if Liberty server was installed by Worklight Server V5.0.6)	No	No	No	No	Stopped (all instances)	Uninstalled

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 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 363.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	No	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 runtime environment for Worklight Server V6.2.0

In these postinstallation 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 Worklight runtime environments to an application server. You must perform these steps for each Worklight runtime environment that you deployed and that you want to upgrade to V6.2.0. If you migrated a Worklight runtime environment 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 Worklight Server runtime environments. 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, whether networked or not, you must stop all instances of Worklight Server runtime environments before you move on to the next steps.

Note: You must do so even if you installed only one Worklight Server runtime environment.

This step is mandatory because in step "Upgrade the Worklight and Worklight Reports databases" on page 383, you upgrade the schema of the databases so that it can be used with Worklight Server V6.2.0. No database schema can be upgraded while Worklight Server runtime environment is running.

After the database is upgraded, only upgraded Worklight Server runtime environments can run, because only upgraded applications can read and write to the new databases. Otherwise, the database might be corrupted.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	See the installation instructions for the fix pack	See the installation instructions for the interim fix	Upgraded	Stopped (all instances)

Important: In addition to stopping the Worklight Console, if you upgrade from IBM Worklight Foundation 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 runtime environment for Worklight Server V6.2.0" on page 379 and prevents the new Worklight runtime environment 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, click 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**.
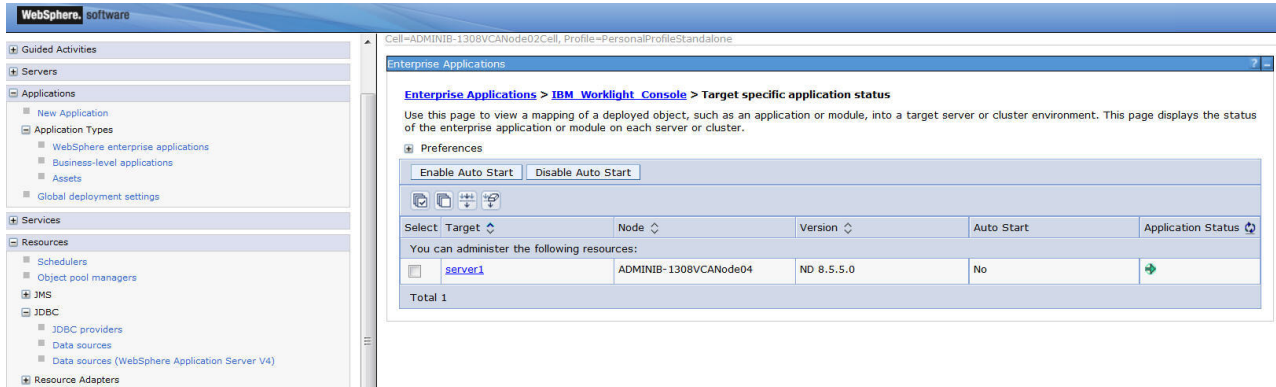
Figure 25.

Shutting down the application server to be upgraded

For certain configurations, in this step you shut down the application server before completing subsequent steps.

About this task

For certain types of application servers (see the following table and "Procedure" section), 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 might fail.

Installation or upgrade of Worklight Server Administration Services

As part of the Worklight Server upgrade, you must install the Administration Services, and optionally the Worklight Console.

The following table lists the upgrade paths in which this step is mandatory.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	• Embedded Liberty, Liberty on Windows, Tomcat on Windows: Stopped • Other application servers: Upgraded	Stopped (all instances)

The procedure is different depending on whether you upgrade from V6.1.0 to V6.2.0 or from V6.2.0 to a fix pack or interim fix.

For an upgrade from V6.1.0 or earlier to V6.2.0
> Follow the steps in "Installing and configuring Worklight Server administration" on page 104.

For an upgrade from V6.2.0 to a fix pack or an interim fix
> 1. Find the Ant file that you created in "Identify the Worklight WAR file and prepare the Ant deployment script" on page 361.
> 2. Make sure that the `taskdef` for the `worklight-ant-deployer.jar` file uses the correct directory. The correct directory is the directory that contains the upgraded installation of Worklight Server V6.2.0.
> 3. Set the **ANT_HOME** environment variable to `/tools/apache-ant-1.8.4/`.
> 4. Run the **minimal-admupdate** target of the Ant file:
>   `<WORKLIGHT_INSTALL_DIR>/tools/apache-ant-1.8.4/bin/ant -f <your_file> minimal-admupdate`

Back up the Worklight and Worklight Reports databases

Back up the contents of your Worklight project databases.

About this task

Important: Before performing this step, verify that you have completed step "Stop all Worklight Server instances" on page 380, and that no instance of Worklight Server is still running, and thus still using these databases.

During the upgrade process in the steps "Upgrade the Worklight and Worklight Reports databases" and "Upgrade the Worklight Server runtime environment" on page 386, the data that is specific to administration and runtime environments are split into distinct databases:

- The Worklight data that is related to administration is moved to the administration database.
- The Worklight and the Worklight Reports runtime databases that you previously updated are migrated to a schema compatible withWorklight Server V6.2.0.

The previous operations cannot be undone.

If, for some reason, you decide to roll back the upgrade of Worklight Server, you 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 installation 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

You must move the Worklight data that is related to administration to the administration database. You must also upgrade the Worklight and the Worklight Reports runtime databases to a schema that is compatible with Worklight Server V6.2.0.

Before you begin

This procedure explains how to upgrade the database with Ant tasks. For a manual upgrade of the databases, see "Manually upgrading the Worklight Server V6.2.0 runtime databases" on page 389 instead.

About this task

Important: First, make sure that you complete step "Stop all Worklight Server instances" on page 380, and that no instance of Worklight Server is still running, and therefore is still using these databases. You must also make sure that you complete step "Installation or upgrade of Worklight Server Administration Services" on page 382, and that the administration database exists.

In this step, you run Ant scripts to perform operations on your Worklight Server 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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	• Embedded Liberty, Liberty on Windows, Tomcat on Windows: Stopped • Other application servers: Upgraded	Stopped (all instances)

Procedure

1. If you upgrade from IBM Worklight Foundation 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 380.
2. Locate the Ant file that you created in section "Identify the Worklight WAR file and prepare the Ant deployment script" on page 361.
3. Verify that the `taskdef` for the `worklight-ant-deployer.jar` uses the correct directory that contains the upgraded installation of Worklight Server V6.2.0.

   In this example, check the value of the **worklight.server.install.dir** property because this property defines 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>
```

> **Important:** This verification step defines the version of IBM Worklight that you use to upgrade 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 version of Apache Ant is the one for which the Worklight deployment scripts are tested. If you do not set this environment variable before you run the script if and have another installation of Ant on your computer, that installation might be used.

5. In the Ant file, make sure that the Ant task `<configuredatabase kind="Worklight">` contains an `<admindatabase>` subelement.

   > **Note:** In the following example code, DB2 is the DBMS.

   ```
   <configuredatabase kind="Worklight">
       <db2 database="WRKLGHT" server="proddb.example.com"
            user="wl6admin" password="wl6pass" schema="WLRT">
         <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>
       <admindatabase runtimeContextRoot=${contextRoot}>
         <db2 database="WLADMIN" server="proddb.example.com"
              user="wl6admin" password="wl6pass" schema="ADMIN">
         </db2>
         <driverclasspath>
           <fileset dir="/opt/database-drivers/db2-9.7">
             <include name="db2jcc4.jar"/>
             <include name="db2jcc_license_*.jar"/>
           </fileset>
         </driverclasspath>
       </admindatabase>
   </configuredatabase>
   ```

   The **${contextRoot}** property contains the value of the context root of the Worklight project.

6. Start 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 start is `<configuredatabase>`.

7. If you use push notifications and you want to upgrade from V6.1.0 or earlier, configure your runtime database manually for push notifications by following the instructions in "Runtime database configuration for Push notifications" on page 393.

This procedure loads the administration database and upgrades the database schemas for the Worklight and Worklight Reports databases to version V6.2.0.

Upgrade the Worklight Server runtime environment

In this step, you run the Ant script to upgrade Worklight runtime environment to V6.2.0. You must repeat this procedure as many times for each runtime environment to upgrade.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	Yes	Yes	• Embedded Liberty, Liberty on Windows, Tomcat on Windows: Stopped • Other application servers: Upgraded	Note: • Embedded Liberty, Liberty on Windows, Tomcat on Windows: Stopped • Other application servers: Partially upgraded

Procedure

1. Locate the Ant file that you created in section "Identify the Worklight WAR file and prepare the Ant deployment script" on page 361.
2. Verify that the `taskdef` for the `worklight-ant-deployer.jar` uses the correct directory containing the upgraded installation of Worklight Server V6.2.0.

   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 Foundation that you use to migrate the databases, to deploy the WAR file, and to install the Worklight runtime library for the Worklight Console.

3. Verify that the **environmentID** attribute for the Worklight runtime environment matches the **environmentID** attribute that you used to install the Worklight Server administration file. For more information, see "Identify the Worklight WAR file and prepare the Ant deployment script" on page 361.

4. Set the environment variable ANT_HOME to *<Worklignt_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. Select the Ant target.
   - To upgrade from V5.0.6.x,, use **install**.
   - To upgrade from V6.0.0.x, or V6.1.0.x, use **uninstall**, and then **install**.
   - To upgrade from V6.2.0 to a fix pack or interim fix, use:
     - Either **uninstall**, then **install** or **minimal-update**.
     - Or **minimal-update**.

   Your choice depends on the nature of the changes in the fix. For more information, see the installation instructions for the fix pack or interim fix.

6. Run Ant with the selected target:

   <Worklignt_Install_Dir>/tools/apache-ant-1.8.4/bin/ant -f *<your file>* *<target defined in step*

   This script has the following effects:
   - It migrates the WAR file to match the runtime environment of the Worklight Server installation.
   - It installs the Worklight runtime environment to the application server, with the root context as defined in the Ant file.

     Note: The context root of the runtime environment cannot be changed because the mobile applications that you previously deployed keep pointing to this context root.
   - It connects the runtime environment to the Administration Services through a Java Management Extensions (JMX) mechanism.
   - If you choose to install the Worklight Console, it connects the Worklight Console to the Administration Services to manage the Worklight runtime environment.

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 363.

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 V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	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 (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 runtime environments.
   - For the JDBC data source:
     - The Worklight runtime and reports 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 andWorklight Server are on the same application server	
V5.0.6.x to V6.2.0	V6.0.0.x to V6.2.0	V6.1.0.x to V6.2.0	V6.2.0 to V6.2.0.x (Fix Pack)	V6.2.0 to interim fix	Application Center Status	Worklight Server Status
Yes	Yes	Yes	See fix pack installation instructions	See interim fix installation instructions	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 Server is upgraded. All applications that you previously deployed are available, along with their environments (those that are supported by Worklight Server V6.2.0).

The URL of the Worklight Console changed. If you did not specify a context root in the Ant file, its context root is `/worklightconsole`.

Additional Worklight Server V6.2.0 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 6.2.0 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 382.
4. Restore the application server. For more information, see "Back up your application server" on page 371.
5. If the server fails to start and load the applications, delete `workarea` of the server 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 installing the Worklight Server administration during the upgrade

You can manually install the Worklight Server administration as part of the Worklight Server upgrade.

Since IBM Worklight Foundation V6.2.0, you must install the Worklight Server administration components. To manually set up the Worklight Server administration, follow the steps detailed in "Manually installing Worklight Server administration" on page 127.

Manually upgrading the Worklight Server V6.2.0 runtime databases

Follow these instructions to manually update the Worklight project databases. The Worklight Server administration environment must be set up before you perform this procedure.

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 the Worklight Reports (by default WLREPORT) and Application Center (by default APPCNTR) databases is done by running a sequence of database scripts.

In IBM Worklight Foundation V6.2.0, the process of updating the Worklight database (by default WRKLGHT) from V6.1.0 is slightly different as it requires to move the runtime administrative data to the administration database. Before you call the script WorklightServer/databases/upgrade-worklight-61-62-<dbms>.sql, run the data migration tool to update the administration database. Then run the script WorklightServer/databases/upgrade-worklight-61-62-<dbms>.sql to update the WRKLGHT database schema. If you forgot to run the data migration tool before you run the SQL script, WorklightServer/databases/upgrade-worklight-61-62-<dbms>.sql is likely to fail, and indicates that the data migration was not run. In that case, the WRKLGHT schema is not updated, or is messed up.

The data migration tool and the database upgrade scripts are both contained in the just-installed (version 6.2.0) Worklight Server directory.

Procedure

For the WRKLGHT database:
1. Apply the proper SQL scripts to upgrade the database to IBM Worklight V6.1.0.
2. Comment the line that refers to the index I_USERPRF_USERID if it was already created.
3. Run the data migration tool.
4. Apply the proper SQL script to upgrade the database from V6.1.0 to V6.2.0.

For the WLREPORT and APPCNTR databases: Apply the proper SQL scripts to upgrade the database to IBM Worklight Foundation V6.2.0.

See the following paragraphs to get information about the index I_USERPRF_USERID, the data migration tool, and the SQL scripts.

Running the data migration tool

Before you run the tool in command line, make sure that the library worklight-ant-deployer.jar is in your current directory, or that your CLASSPATH variable references the directory it is in. Example on a Unix/Linux machine:

```
# Go the directory library  worklight-ant-deployer.jar is in
$ cd $WL_INSTALL_DIR/WorklightServer
# Print the usage
$ java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration62.MigrationTool usage
Usage:
        java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration62.MigrationTool [op

Options:
  -p <project>      The name of the project to create.
  -sourceurl        The path to the source database.
  -sourceschema     The name of the schema of the source database.
  -sourcedriver     The fully qualified driver class name of the source
                    database. This driver must be in the class path.
  -sourceuser       The user name of the source database.
  -sourcepassword   The password of the source database.
```

```
-sourceproperty <key> <value>   Adds additional OpenJPA properties to the connection
                      of the source database.
-targeturl          The path to the target database.
-targetschema       The name of the schema of the target database.
-targetdriver       The fully qualified driver class name of the target
                      database. This driver must be in the class path.
-targetuser         The user name of the target database.
-targetpassword     The password of the target database.
-targetproperty <key> <value>   Adds additional OpenJPA properties to the connection
                      of the target database.
```

```
# Example
$ java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration62.MigrationTool
  -p worklight
  -sourceurl jdbc:db2://proddb.example.com:50000/WRKLGHT
  -sourceschema WLRT
  -sourcedriver com.ibm.db2.jcc.DB2Driver
  -sourceuser wluser1
  -sourcepassword wluser1_pswd
  -targeturl jdbc:db2://proddb.example.com:50000/WLADMIN
  -targetschema ADMIN
  -targetdriver com.ibm.db2.jcc.DB2Driver
  -targetuser wluser2
  -targetpassword wluser2_pswd
```

Note:
- The name of the project to create (worklight in the example) must be the context root where the Worklight runtime component (project WAR file) is deployed. For example, if the runtime component is deployed in the application server with a context root /worklight, then the project name must be worklight. The applications will be assigned to this project name. When a runtime component starts, it contacts the administration service to get the applications and the adapters it needs to serve. The runtime component uses its project name, computed by removing the initial / from its context root, to indicate which applications it needs. If the project name is not the same as the context root without the initial slash, then the migrated applications and the adapters are not visible by the runtime component.
- For MySQL databases, the schema options '-sourceschema' and '-targetschema' must be left unspecified. The name of the schema to use will be the name of the database specified in the connection URL.

SQL scripts

Important: Before you apply the SQL scripts that upgrade the WRKLGHT database from V6.1.0 to V6.2.0, you must check whether the index I_USERPRF_USERID, in column USER_ID of the table GADGET_USER_PREF, exists. If it does, in the WorklightServer/databases/upgrade-worklight-61-62-<*dbms*>.sql script, comment the following line that refers to its creation before running it:

```
INDEX I_USERPRF_USERID ON GADGET_USER_PREF (USER_ID);
```

For more information about this index, see the technote IBM Worklight queries on the GADGET_USER_PREF table might take time.

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)

For an upgrade from Worklight Server V6.1.0.x to V6.2.0:
- WorklightServer/databases/upgrade-worklight-61-62-db2.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-61-62-db2.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-61-62-db2.sql (for APPCNTR)

These scripts are applied similarly to steps 4 and 6 in "Setting up your DB2 databases manually" on page 1121

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)

For an upgrade from Worklight Server V6.1.0.x to V6.2.0:
- WorklightServer/databases/upgrade-worklight-61-62-mysql.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-61-62-mysql.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-61-62-mysql.sql (for APPCNTR)

These scripts are applied similarly to step 1.b in "Setting up your MySQL databases manually" on page 1131.

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)

For an upgrade from Worklight Server V6.1.0.x to V6.2.0:

- `WorklightServer/databases/upgrade-worklight-61-62-oracle.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-61-62-oracle.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-61-62-oracle.sql` (for APPCNTR)

These scripts are applied similarly to step 3 in "Setting up your Oracle databases manually" on page 1136.

Runtime database configuration for Push notifications

Note: The following section applies if you use push notifications and you upgrade from V6.1.0 or earlier.

To ensure that push notifications work as expected, you must complete some additional manual configuration. After your migration to V6.2.0 is complete (this includes a complete migration of both the runtime database and the runtime environment) and the application server is restarted, complete the steps based on your database type. No changes are required if you use MySQL or Derby.

This task requires administrator access privileges.

DB2

1. Change *SCHEMANAME* instances to actual names.
2. Replace X and Y values based on the given description.

   ```
   SELECT MAX(ID) FROM SCHEMANAME.PUSH_DEVICES;
   SELECT MAX(ID) FROM SCHEMANAME.PUSH_SUBSCRIPTIONS;

   --Value of X = Result of the selected query of PUSH_DEVICES + 1. For example, if SELEC
   --SCHEMANAME.PUSH_DEVICES returns 100, then X = 101;
   --Value of Y = Result of the selected query of PUSH_SUBSCRIPTIONS + 1. For example, if
   --SCHEMANAME.PUSH_SUBSCRIPTIONS returns 100, then Y = 101;
   ```

```
                    ALTER SEQUENCE SCHEMANAME.PUSHDEVICE_SEQ RESTART WITH X ;
                    ALTER SEQUENCE SCHEMANAME.PUSHSUBSCRIPTION_SEQ RESTART WITH Y ;

                    VALUES NEXT VALUE FOR SCHEMANAME.PUSHDEVICE_SEQ;
                    VALUES NEXT VALUE FOR SCHEMANAME.PUSHSUBSCRIPTION_SEQ;
```

Oracle

1. Change *SCHEMANAME* instances to actual names.

2. Replace X and Y values (based on the given description) while running
   the query.

```
SELECT MAX(ID) FROM SCHEMANAME.PUSH_DEVICES;
SELECT MAX(ID) FROM SCHEMANAME.PUSH_SUBSCRIPTIONS;

SELECT LAST_NUMBER FROM ALL_SEQUENCES WHERE SEQUENCE_NAME='PUSHDEVICE_SEQ';
SELECT LAST_NUMBER FROM ALL_SEQUENCES WHERE SEQUENCE_NAME='PUSHSUBSCRIPTION_SEQ';

--Take note of the resulting value of each query above to use in the following X and Y ca

--Value of X = (MAX(ID) OF PUSH_DEVICES - LAST_NUMBER OF PUSHDEVICE_SEQ) + 20. For exampl
--SCHEMANAME.PUSH_DEVICES returned 100 and SELECT LAST_NUMBER FROM ALL_SEQUENCES
--(where SEQUENCE_NAME='PUSHDEVICE_SEQ' is 50), then X=100-50+20 = 70

--Value of Y = (MAX(ID) OF PUSH_SUBSCRIPTIONS - LAST_NUMBER OF PUSHSUBSCRIPTION_SEQ) + 20
--MAX(ID) FROM SCHEMANAME.PUSH_SUBSCRIPTIONS returned 100, then Y = 101 and SELECT --LAST
--FROM ALL_SEQUENCES (where SEQUENCE_NAME='PUSHSUBSCRIPTION_SEQ' is 50), then Y=100-50+20

ALTER SEQUENCE SCHEMANAME.PUSHDEVICE_SEQ INCREMENT BY X ;
SELECT SCHEMANAME.PUSHDEVICE_SEQ.NEXTVAL FROM dual;
ALTER SEQUENCE SCHEMANAME.PUSHDEVICE_SEQ INCREMENT BY 1;

ALTER SEQUENCE SCHEMANAME.PUSHSUBSCRIPTION_SEQ INCREMENT BY Y ;
SELECT SCHEMANAME.PUSHSUBSCRIPTION_SEQ.NEXTVAL FROM dual;
ALTER SEQUENCE SCHEMANAME.PUSHSUBSCRIPTION_SEQ INCREMENT BY 1;
```

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 375 and "Upgrade the
Worklight Server runtime environment" on page 386, use the following procedure:

- Upgrade the databases manually as specified in section "Manually upgrading
  the Worklight Server V6.2.0 runtime databases" on page 389.
- Review the manual installation procedures at:
  - "Manually installing Application Center" on page 243
  - "Deploying a project WAR file and configuring the application server
    manually" on page 1163
- 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 runtime environment, 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 1164.

Verifying and updating the HTTP redirections for Worklight Server V6.2.0

If you are upgrading Worklight Server on a clustered application server environment, you should also update IBM HTTP Server after you install IBM Worklight Foundation V6.2.0.

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 installWorklight Server V6.2.0. For general information about installing these types of application server, see:

- "Setting up IBM Worklight Foundation in an IBM WebSphere Application Server Network Deployment V8.5 cluster environment" on page 305
- "Setting up an IBM HTTP Server in an IBM WebSphere Application Server Liberty profile farm" on page 316

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>
```

Updating DB2 schema names in the case of a manual installation

In this step, you must update the DB2 schema if you are using DB2 for the Worklight or Worklight reports databases.

About this task

For the Worklight and Worklight reports databases, IBM Worklight V6.1.0 expects double quotation marks around the schema name. However, IBM Worklight Foundation V6.2.0 expects the schema name without surrounding double quotation marks.

Procedure

Perform one of the following steps, based on your installation.

1. WebSphere Application Server Liberty profile:
   a. Edit the `server.xml` file in the `usr/servers/serverName` directory.
   b. Look for the `<properties.db2.jcc .../>` element in the `<dataSource jndiName="contextroot/jdbc/WorklightDS" ...>` and `<dataSource jndiName="contextroot/jdbc/WorklightReportsDS" ...>` elements.

c. Remove the double quotation marks around the value of the `currentSchema` attribute, if this attribute is present. For example, change `currentSchema='"wrkschem"'` or `currentSchema=""wrkschem""` to `currentSchema='wrkschem'`.

2. WebSphere Application Server full profile:

   a. Sign in to WebSphere Application Server administrative console.

   b. Click **Resources** > **JDBC** > **Data sources**

   c. For each database with the JNDI name `jdbc/WorklightDS` or `jdbc/WorklightReportsDS`, possibly with a suffix:

      1) Select the data source

      2) Click **Additional properties** > **Custom properties**.

      3) Select the `currentSchema` property.

      4) If the value is not empty, remove the double quotation marks around the value. For example, change the value `"wrkschem"` to `wrkschem`.

      5) Click **OK**.

      6) Click **Save** to save the changes.

3. Tomcat:

   a. Edit the `server.xml` file in the `conf` directory.

   b. In the `<Resource name="jdbc/WorklightDS" .../>` and `<Resource name="jdbc/WorklightReportsDS" .../>` elements, remove the double quotation marks around the value of the `currentSchema` connection property in the `url` attribute, if this property is present. For example, change `url='jdbc:db2://dbserver.example.com:50000/WRKLGHT:currentSchema="wrkschem";'` to `url='jdbc:db2://dbserver.example.com:50000/WRKLGHT:currentSchema=wrkschem;'`.

Updating deployment scripts

If you use Ant tasks `<app-deployer>` or `<adapter-deployer>` to deploy apps or adapters, you must update the Ant scripts to use the Ant task `<wladm>`.

If you have Ant scripts that deploy apps or adapters by using the Ant tasks `<app-deployer>` or `<adapter-deployer>`, you must update them to use the Ant task `<wladm>`. The Ant tasks `<app-deployer>` or `<adapter-deployer>` no longer apply in IBM Worklight Foundation V6.2.0.

In the initialization commands of the Ant script, replace

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

with

```
<taskdef resource="com/worklight/ant/deployers/antlib.xml">
  <classpath>
    <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

Replace `<app-deployer>` invocations

```
<app-deployer deployable="myApp.wlapp"
              worklightserverhost="http://server-address:port/project-name"
              userName="username" password="password"/>
```

with

```
<wladm url="https://server-address:secure-port/worklightadmin" user="username" password="password"
  <deploy-app runtime="project-name" file="myApp.wlapp"/>
</wladm>
```

where for *worklightadmin*, you need to substitute the actual context root of the
Worklight administration services web application. For *username* and *password*, you
need to pick a user that is in the role **worklightadmin** or **worklightdeployer**.

Replace `<adapter-deployer>` invocations

```
<adapter-deployer deployable="myAdapter.adapter"
                  worklightserverhost="http://server-address:port/project-name"
                  userName="username" password="password"/>
```

with

```
<wladm url="https://server-address:secure-port/worklightadmin" user="username" password="password"
  <deploy-adapter runtime="project-name" file="myAdapter.adapter"/>
</wladm>
```

where for *worklightadmin*, you need to substitute the actual context root of the
Worklight administration services web application. For *username* and *password*, you
need to pick a user that is in the role **worklightadmin** or **worklightdeployer**.

For more information about `<wladm>`, see "Administering Worklight applications
through ANT" on page 1258.

Upgrading from Worklight Server V6.2.0 to V6.2.0.x in a production environment

Some upgrade steps are optional when you upgrade to a fix pack or an interim fix.
You must follow specific instructions when you upgrade from Worklight Server
V6.2.0 to V6.2.0.x.

To upgrade from Worklight Server V6.2.0 to V6.2.0.x. (fix pack or interim fix) in a
production environment, you must follow the steps that are listed in the topic
"Upgrading to Worklight Server V6.2.0 in a production environment" on page 355,
but some of these steps are optional for a fix pack or an interim fix. You can see in
Table 1 the status of these optional steps for the upgrade to Worklight Server
V6.2.0.x.

Table 87. Status of the optional steps for upgrading to a V6.2.0 fix pack or interim fix

Upgrade step	Is this step required for V6.2.0.x fix pack?
"Review and note the Application Server configuration for Worklight Server and Application Center" on page 363	Not required
"Stop all instances of the Application Center applications" on page 374	Required
"Back up the Application Center database" on page 374	Not required
"Restore the Application Center configurations and restart the application server" on page 378	Only restart is required
"Stop all Worklight Server instances" on page 380	Required

Table 87. Status of the optional steps for upgrading to a V6.2.0 fix pack or interim fix (continued)

Upgrade step	Is this step required for V6.2.0.x fix pack?
"Installation or upgrade of Worklight Server Administration Services" on page 382	Use the Ant target `minimal-admupdate`
"Back up the Worklight and Worklight Reports databases" on page 382	Not required
"Upgrade the Worklight and Worklight Reports databases" on page 383	Not required
"Upgrade the Worklight Server runtime environment" on page 386	Use the Ant target `minimal-update` (step 5)
"Restore the Worklight Server Configuration" on page 387	Not required
"Restart the application server" on page 388	Required

Rolling upgrade procedure to apply a fix pack to IBM Worklight Foundation V6.2.0

You can perform a rolling upgrade to apply a fix pack or an interim fix to an installation of Worklight Server V6.2.0 without downtime of the Worklight runtime environment. Performing this rolling upgrade ensures that there is no interruption of service for the mobile applications that query the Worklight Server .

To perform this rolling upgrade, you must install the upgraded version of IBM Worklight Foundation in a different environment, for example a new cluster in WebSphere Application Server Network Deployment. This environment must be connected to the same databases as the initial IBM Worklight Foundation installation. You must then switch HTTP traffic progressively from the old environment to the new environment.

The procedure for an in-place upgrade, with interruption of service, is documented at Chapter 8, "Upgrading to IBM Worklight Foundation V6.2.0," on page 343.

Important: This procedure applies to a Worklight Server installation, including the Administration Services, the Worklight Console , and the Worklight runtime environment, but does not apply to the Application Center.

The following topics explain what you must plan for your production environment, and the steps of the rolling upgrade procedure for IBM Worklight Foundation, installed in one cluster in IBM WebSphere Application Server, with the HTTP traffic routed by an IBM HTTP Server, and web server plug-ins for IBM WebSphere Application Server.

Planning the rolling upgrade procedure

You must plan the steps of the rolling upgrade procedure to install a fix pack to IBM Worklight Foundation without downtime.

You must review, adapt, and test this upgrade procedure for your production environment. Other components that interact with IBM Worklight Foundation in your production environment might require extra steps or changes to that procedure.

The goal of this procedure is to switch HTTP-based traffic from one installation of Worklight Server to an upgraded installation of Worklight Server. This is done without losing any data in the Worklight databases, and without visible impact for users of Worklight applications that have an active session while this procedure is applied.

If other protocols than HTTP are used by some components of your application to interact with the Worklight Server, then you must include in your upgrade plan a way to route that traffic during a rolling upgrade procedure.

For example, if you use a pull mechanism for push notifications, you must review the risk of double notification or lost notification in a rolling upgrade procedure. For more information, see "Possible Worklight push notification architectures" on page 842.

You must review, with the development team, the code of all your adapters to identify extra steps that might be required for a rolling upgrade procedure. This is the case especially if these adapters require external resources, or use other communication protocols than HTTP, such as JMS or WebSphere MQ.

During the rolling upgrade, you must also stop management operations, such as uploading a new application or a new adapter to the console. If a new version of a Worklight application or adapter is uploaded to the Worklight Server during the upgrade procedure, some servers on the clusters might not be notified of this change and might continue to operate with the old artifact. If a new Worklight application is uploaded to the Worklight Server and the runtimes do not all have the same version, it might trigger arbitrary direct update sessions for the users of Worklight apps, depending on the server to which they were routed. Identify all Worklight administration users that have a privilege to upgrade an application. Their role is defined as `worklightadmin` or `worklightdeployer`. You can then notify them of the beginning and the end of the rolling upgrade. During that period, they must not upload any adapter or application.

This procedure requires to temporarily duplicate the environment. You might find it convenient to apply this procedure at a period of low traffic, so that you can use existing hardware resources for the servers of the new environment. You must double the number of WebSphere application servers during the rolling upgrade, and the hardware must have enough memory to run these servers without paging. The CPU requirements must not increase significantly during that procedure because the use on the servers that are being brought online would ramp up as new sessions get routed to the new version of the application. The CPU use on the servers that run the old version of the application must ramp down as existing sessions end.

Overview of the rolling upgrade procedure

Learn about the steps of a rolling upgrade procedure.

You must perform the following actions to complete a rolling upgrade procedure. These steps are detailed in the following topics.
- Stop management operations while you apply the rolling upgrade procedure. No management operation, such as uploading a new application version or a new adapter, must be performed during a rolling upgrade.
- Duplicate the application server environment and install the fix pack IBM Worklight Foundation V6.2.0.x in that duplicated environment, for example in a

new cluster. You must use the existing Administration database, Worklight runtime database, and Worklight reports database.

- Start a server in the duplicated environment.
- Direct some of the new HTTP sessions to the new servers and drain the servers of the previous installation so that they do not receive new HTTP sessions.

    Note: IBM Worklight Foundation uses session affinity and locally stores data about the state of sessions in a server. When routing traffic to the new cluster, existing sessions must continue to be routed to the old server with which they started.

- When the old servers are all drained and there is no longer any active session, and the new Worklight Server is confirmed to work correctly, shut down the old servers. If required, uninstall the old IBM Worklight Foundation version from those servers.
- When the old environment is shut down, you can authorize management operations again.

Performing a rolling upgrade to install a fix pack

Learn how to perform a rolling upgrade to install a fix pack to IBM Worklight Foundation, assuming the following topology: IBM Worklight Foundation is installed in one cluster in IBM WebSphere Application Server, and the HTTP traffic is routed by an IBM HTTP Server and web server plug-ins for IBM WebSphere Application Server.

About this task

The following topics present the steps of the rolling upgrade procedure, in the order in which they must be completed.

Note: The cluster in which IBM Worklight Foundation is installed is called `cluster_WL61` in the following topics.

Stopping management operations

Managements operations must not be performed during a rolling upgrade.

Procedure

You must ensure that all management operations, such as uploading a new application or a new adapter, are stopped while you perform the rolling upgrade procedure.

Notify users with the privilege `worklightadmin` or `worklightdeployer` that they cannot deploy any Worklight artifact until the upgrade procedure is complete.

Installing the IBM Worklight Foundation fix pack in a new cluster

You must create a new cluster, in which you install the IBM Worklight Foundation fix pack. Here, the procedure targets an installation in a WebSphere Application Server Network Deployment environment, in a single cell, with IBM HTTP Server and web server plug-ins for IBM WebSphere Application Server.

Procedure

1. Create a cluster in IBM WebSphere Application Server. In the rest of this document, this cluster is called `cluster_WL61FP1`.
2. Create servers in this cluster.

3. For each server, set a weight of 0.
4. Install the IBM Worklight Foundation fix pack in this cluster. You can install this fix pack with the IBM Worklight Foundation Ant tasks, or manually.
   - To install this fix pack with the IBM Worklight Foundation Ant tasks, follow the steps 5 - 9 on page 402.
   - To install this fix pack manually, follow the steps 10 on page 402 - 14 on page 403.

Installing the fix pack with the IBM Worklight Foundation Ant tasks:
5. With IBM Installation Manager, install the IBM Worklight Foundation fix pack on the computer where the WebSphere Application Server Deployment Manager is installed.

   Note: Do not install the Application Center.
6. Verify that WebSphere Application Server is not set to automatically generate and propagate the web plug-in. To be sure that the installation of the Worklight fix-pack does not generate and propagate a new web plug-in that you did not review, perform the following steps.
   a. Open the WebSphere Application Server administration console.
   b. Go to **Servers** > **Server Types** > **Web Servers**.
   c. In the table, click the web server.
   d. Under **Additional Properties**, click **Plug-in properties**.
   e. Make sure that the check box for **Automatically generate the plug-in configuration file** is not selected.
   f. Make sure that the check box for **Automatically propagate plug-in configuration file** is not selected.
7. Copy the Ant file that you used to install IBM Worklight Foundation in the cluster `cluster_WL61`.
   a. Modify the cluster name, for example ${was.nd.cluster.name} in the code example in step 8.
   b. Modify the environment ID.

      You use this environment ID to distinguish the Administration Services and Worklight runtime environment from the two clusters. The new ID also generates different application names to avoid name conflicts in the WebSphere Application Server cell.
   c. You must make these modifications for the following Ant tasks. Use the same environment ID in all the tasks.
      - `configureapplicationserver`
      - `updateapplicationserver`
      - `unconfigureapplicationserver`
      - `installworklightadmin`
      - `updateworklightadmin`
      - `uninstallworklightadmin`

      Important: The environment ID determines which instance of Administration Services manages the deployed runtime environments. All runtime environments must have the same environment ID as the Worklight Server administration components.
8. Run the `adminstall` target of the Ant file.

9. Run the `install` target of the Ant file that installs the Worklight runtime environment. If you have more than one Worklight runtime environment, repeat this operation for all of them:

   Note: Do not change the database settings, the context roots, or the other parameters of the installation.

   ```
   <!-- Start of the install target Generated by Worklight -->
     <target name="install">
       <configureapplicationserver environmentId="${worklight.environment.id}" contextroot="${workl
         <project warfile="${worklight.project.war.file}"/>
         <applicationserver>
           <websphereapplicationserver installdir="${appserver.was.installdir}"
                                       profile="${appserver.was.profile}"
                                       user="${appserver.was.admin.name}"
                                       password="${appserver.was.admin.password}">
             <cluster name="${appserver.was.nd.cluster}"/>
           </websphereapplicationserver>
         </applicationserver>
   ```

Installing the fix pack manually:

10. Install the data sources that point to the administration and Worklight runtime databases in the cluster.
    - For instructions about the administration database, see the following documentation:
      – "Configuring WebSphere Application Server for DB2 manually for Worklight Server administration" on page 129
      – "Configuring WebSphere Application Server for Oracle manually for the Worklight Server administration" on page 140
    - For instructions about the runtime database, see the following documentation:
      – "Configuring WebSphere Application Server for DB2 manually" on page 1124
      – "Configuring WebSphere Application Server for Oracle manually" on page 1138

11. Select a name for an environment ID that is used for all the web applications that you installed in step 12.

    For example, FP1.

    Note: You must not have any other installation of IBM Worklight Foundation in the WebSphere Application Server cell that is using the same environment ID.

12. Install the Administration Services as documented in "Configuring WebSphere Application Server for Worklight Server administration manually" on page 144, with the following change:

    In **Environment entries for Web modules** for the Administration Services, set the value of `ibm.worklight.admin.environmentid` to the environment ID that you selected in step 11.

13. Install the Worklight runtime environment as documented in "Configuring WebSphere Application Server for Worklight Server administration manually" on page 144, with the following change:

    In **Environment entries for Web modules** for the Worklight runtime environment, set the value of `ibm.worklight.admin.environmentid` to the environment ID that you selected in step 11.

14. Use a different name for the WebSphere applications than the one that you used in the first cluster.

Completing the configuration of the new installation of IBM Worklight Foundation

You must complete the configuration of the new installation of IBM Worklight Foundation with security settings, update of the JNDI properties, and other configuration settings.

About this task

This configuration includes the following parameters for the Administration Services application, and for the Worklight Console application. For more information, see "Configuring Worklight Server" on page 181.

Procedure

1. Configure the security settings that define the users for each of the following roles: `worklightadmin`, `worklightdeployer`, `worklightmonitor`, `worklightoperator`.
2. Update the JNDI properties that you modified during the installation in the **WL61FP1** cluster:
   * For the Administration Services and the Worklight Console.
   * For each Worklight runtime environment.

Verifying the new installation of IBM Worklight Foundation

You must make sure that IBM Worklight Foundation is installed properly.

Procedure

1. Start all the servers in the **Worklight61FP1** cluster. This action starts all the Worklight runtime environments. They synchronize with the Administration Services, and download the application and adapter artifacts they need to be ready to serve requests.
2. Log to the Worklight Console. At this stage, IBM HTTP Server must not route traffic to that installation, so you must connect directly to the host name and port of a server. If the Worklight Console is installed with the default context root, which is `worklightconsole`, the URL looks like the following example: `http://<hostname>:<httpPortOfServer>:/worklightconsole/`
3. Verify that the Worklight runtime environments are present and that they do not report any error.
4. Optional: You might also perform an additional smoke test of the adapters that are specific to your IBM Worklight Foundation installation.

Switching progressively the HTTP traffic to the new cluster, with session affinity

You must modify the HTTP plug-in file to route the HTTP traffic to the new installation of IBM Worklight Foundation

Before you begin

The following procedure requires modifications of the HTTP plug-in file, `plugin-cfg.xml`. Before you perform this procedure in production, you must test it in a test environment.

Important: If errors occur during these steps, it would result in incorrect traffic routing and might impact all applications in the cell of the WebSphere Application Server.

About this task

The procedure to route the traffic is based on the following properties of the web plug-in:

- The plug-in routes traffic to a server cluster that is based on the definition of the cluster members in its `<ServerCluster>` listing. The HTTP plug-in has no other information about the target servers other than what is defined in the plug-in configuration file. Even if a collection of servers is defined in a WebSphere cell as being in two separate clusters, they can be defined in one cluster from the point of view of the plug-in. With this property, you can use the plug-in to route traffic between the clusters **cluster_WL61** and **cluster_WL61FP1**.

- The `LoadBalanceWeight` attribute of the `Server` element is used to statically assign a weighting factor that is associated with the round-robin distribution of new requests among the servers that are in a cluster. When this attribute is set to zero, this is a signal to the plug-in to stop sending new requests to that application server. Requests that are associated with existing sessions on that server continue to flow to it, but as those sessions get terminated, the server stops having any active sessions.

- The `plugin-cfg.xml` file is re-read periodically by the plug-in to the HTTP server, with a default refresh interval of 1 minute.

For more information about updating the `plugin-cfg.xml` file, see "Setting up IBM Worklight Foundation in an IBM WebSphere Application Server Network Deployment V8.5 cluster environment" on page 305.

Procedure

1. Progressively, set the weight of a server in the **cluster_WL61** to 0.
2. Move a server of the cluster **cluster_WL61FP1** to the definition of **cluster_WL61**, with a weight of 2.
3. Wait for the server in cluster **cluster_WL61** to drain so that most of the session terminates and that it stops using CPU resources.
4. Repeat the procedure for the next server.

Example

Before you start the procedure:

```
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61">
  <Server CloneID="a8er1kj2" LoadBalanceWeight="2" Name="ServerWL61_1">
    <Transport Hostname="test1.ibm.com" Port="9081" Protocol="http"/>
  </Server>
  <Server CloneID="a8er2kd3" LoadBalanceWeight="2" Name="ServerWL61_2">
    <Transport Hostname="test2.ibm.com" Port="9082" Protocol="http"/>
  </Server>
  <Server CloneID="a8es3as1" LoadBalanceWeight="2" Name="ServerWL61_3">
    <Transport Hostname="test3.ibm.com" Port="9083" Protocol="http"/>
  </Server>
  <PrimaryServers>
    <Server Name="ServerWL61_1"/>
    <Server Name="ServerWL61_2"/>
    <Server Name="ServerWL61_3"/>
  </PrimaryServers>
</ServerCluster>
[...]
```

```
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61FP1">
  <Server CloneID="a8sd1kj2" LoadBalanceWeight="0" Name="ServerWL61FP1_1">
    <Transport Hostname="test1.ibm.com" Port="9084" Protocol="http"/>
  </Server>
  <Server CloneID="a8as2kd3" LoadBalanceWeight="0" Name="ServerWL61FP1_2">
    <Transport Hostname="test2.ibm.com" Port="9085" Protocol="http"/>
  </Server>
  <Server CloneID="a8qa3as1" LoadBalanceWeight="0" Name="ServerWL61FP1_3">
    <Transport Hostname="test3.ibm.com" Port="9086" Protocol="http"/>
  </Server>
  <PrimaryServers>
    <Server Name="ServerWL61FP1_1"/>
    <Server Name="ServerWL61FP1_2"/>
    <Server Name="ServerWL61FP1_3"/>
  </PrimaryServers>
</ServerCluster>
[...]
<!-- Example of the UriGroups and Routes. They are not changed while you switch the traffic -->
<UriGroup Name="prod_vhost_cluster_WL61_URIs">
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklightconsole/*"/>
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklightadmin/*"/>
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklight/*"/>
</UriGroup>
<Route ServerCluster="ClusterX"
  UriGroup="prod_vhost_cluster_WL61_URIs" VirtualHostGroup="default_host"/>
<UriGroup Name="test_vhost_cluster_WL61FP1_URIs">
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklightconsole/*"/>
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklightadmin/*"/>
  <Uri AffinityCookie="JSESSIONID"
      AffinityURLIdentifier="jsessionid" Name="/worklight/*"/>
</UriGroup>
<Route ServerCluster="ClusterY" UriGroup="test_vhost_cluster_WL61FP1_URIs"
  VirtualHostGroup="test_host"/>
```

Moving a server:

```
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61">
  <!-- Server ServerWL61_1 has a weight of 0 -->
  <Server CloneID="a8er1kj2" LoadBalanceWeight="0" Name="ServerWL61_1">
    <Transport Hostname="test1.ibm.com" Port="9081" Protocol="http"/>
  </Server>
  <Server CloneID="a8er2kd3" LoadBalanceWeight="2" Name="ServerWL61_2">
    <Transport Hostname="test2.ibm.com" Port="9082" Protocol="http"/>
  </Server>
  <Server CloneID="a8es3as1" LoadBalanceWeight="2" Name="ServerWL61_3">
    <Transport Hostname="test3.ibm.com" Port="9083" Protocol="http"/>
  <!-- Server ServerWL61F1_1 added to the cluster_WL61 in the plugin-cfg file -->
  <Server CloneID="a8sd1kj2" LoadBalanceWeight="2" Name="ServerWL61FP1_1">
    <Transport Hostname="test1.ibm.com" Port="9084" Protocol="http"/>
  </Server>
  <PrimaryServers>
    <Server Name="ServerWL61_1"/>
    <Server Name="ServerWL61_2"/>
    <Server Name="ServerWL61_3"/>
    <Server Name="ServerWL61FP1_1"/>
  </PrimaryServers>
</ServerCluster>
[...]
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61FP1">
  <Server CloneID="a8as2kd3" LoadBalanceWeight="0" Name="ServerWL61FP1_2">
    <Transport Hostname="test2.ibm.com" Port="9085" Protocol="http"/>
  </Server>
```

```
      </Server>
      <Server CloneID="a8qa3as1" LoadBalanceWeight="0" Name="ServerWL61FP1_3">
        <Transport Hostname="test3.ibm.com" Port="9086" Protocol="http"/>
      </Server>
      <PrimaryServers>
        <Server Name="ServerWL61FP1_2"/>
        <Server Name="ServerWL61FP1_3"/>
      </PrimaryServers>
</ServerCluster>
```

End of the transition:

```
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61">
  <!-- Server ServerWL61_X have a weight of 0 -->
  <Server CloneID="a8er1kj2" LoadBalanceWeight="0" Name="ServerWL61_1">
    <Transport Hostname="test1.ibm.com" Port="9081" Protocol="http"/>
  </Server>
  <Server CloneID="a8er2kd3" LoadBalanceWeight="0" Name="ServerWL61_2">
    <Transport Hostname="test2.ibm.com" Port="9082" Protocol="http"/>
  </Server>
  <Server CloneID="a8es3as1" LoadBalanceWeight="0" Name="ServerWL61_3">
    <Transport Hostname="test3.ibm.com" Port="9083" Protocol="http"/>
  <!-- Server ServerWL61F1_X added to the cluster_WL61 in the plugin-cfg file -->
  <Server CloneID="a8sd1kj2" LoadBalanceWeight="2" Name="ServerWL61FP1_1">
    <Transport Hostname="test1.ibm.com" Port="9084" Protocol="http"/>
  </Server>
  <Server CloneID="a8as2kd3" LoadBalanceWeight="2" Name="ServerWL61FP1_2">
    <Transport Hostname="test2.ibm.com" Port="9085" Protocol="http"/>
  </Server>
  </Server>
  <Server CloneID="a8qa3as1" LoadBalanceWeight="2" Name="ServerWL61FP1_3">
    <Transport Hostname="test3.ibm.com" Port="9086" Protocol="http"/>
  </Server>
  <PrimaryServers>
    <Server Name="ServerWL61_1"/>
    <Server Name="ServerWL61_2"/>
    <Server Name="ServerWL61_3"/>
    <Server Name="ServerWL61FP1_1"/>
    <Server Name="ServerWL61FP1_2"/>
    <Server Name="ServerWL61FP1_3"/>
  </PrimaryServers>
</ServerCluster>
[...]
<ServerCluster LoadBalance="Round Robin" Name="cluster_WL61FP1">
  <PrimaryServers>
  </PrimaryServers>
</ServerCluster>
```

Performing a rollback procedure

You might want to perform a rollback procedure to restore your initial
configuration if a problem occurs.

Procedure

If a problem is detected while you are switching the traffic, you must restore the
plugin-cfg.xml to its initial state so that the HTTP traffic is routed again to the
initial installation of IBM Worklight Foundation.

Uninstalling IBM Worklight Foundation from the old cluster

You must uninstall IBM Worklight Foundation from the cluster that it was
previously installed in, and update the appropriate settings accordingly.

Procedure

1. When the migration is complete and the sessions are stopped, shutdown the `cluster_WL61` cluster.

2. You can allow management operations to start again and notify users with privilege `worklightadmin` or `worklightdeployer` that they are allowed to deploy Worklight artifacts because the upgrade procedure is complete.

3. Uninstall IBM Worklight Foundation from the cluster `cluster_WL61`.

4. Update the `plugin-cfg.xml` so that it no longer references the old cluster `cluster_WL61`.

5. Verify that the HTTP traffic is routed correctly to the new cluster. For example, you can activate the log file of the web plug-in, and review the log.

Chapter 9. Developing Worklight applications

You use Worklight Studio, the 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 IBM Worklight Foundation. 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 Foundation 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 Foundation 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, Objective-C, or C#), and IBM Worklight Foundation 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 Worklight adapters. With integrated development tools, you can develop and test these Worklight 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.

Startup behavior

Every project has an associated WAR file. In Worklight Studio, you deploy the WAR file to Worklight Development Server during standard development

activities. Worklight Studio remembers the last deployed project to the server. When Worklight Studio restarts, all deployed Worklight WAR applications are deleted from the server, except from the last one. The behavior is to avoid a Timeout error with the Worklight Development Server during server startup in case there are many WAR applications deployed.

Note: You can inhibit this behavior and make it work exactly like IBM Worklight V6.1 by using the following steps:

- Close Worklight Studio.
- Go to the following path: `/<workspace_path>/.metadata/.plugins/ org.eclipse.core.runtime/.settings`
- Open `com.worklight.studio.plugin.prefs` file with a text editor.
- Add **`com.worklight.studio.plugin.avoidServerCleanup=true`**.
- Save the file and restart Worklight Studio.

Artifacts produced during development cycle

When you use Worklight Studio within the 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 a 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 Worklight developer (for example, retrieve data from a remote database). Adapter code is accessed by 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 1105.

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.

Worklight projects

To develop your mobile applications with IBM Worklight Foundation, you must first create a project in 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 430.
- A `build-settings.xml` file, for you to prepare minification and concatenation configurations for each environment. For more information about this file, see "Worklight application build settings" on page 723.

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.

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.

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 Worklight projects

You use Worklight Studio to create a Worklight project.

About this task

With Worklight Studio, you create a Worklight project as a place where you develop your apps. When you create a 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 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 Worklight API for integration, security, and application management.

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

Procedure

To create a Worklight project and a first app in it:
1. Select **File** > **New** > **Worklight Project**.
2. In the **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 "Creating hybrid applications with IBM Worklight Application Framework" on page 559.
`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 a Worklight project

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

About this task

WithWorklight Studio, you can create and develop a Worklight application in an existing 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 "Creating hybrid applications with IBM Worklight Application Framework" on page 559.
     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** 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**, **Java ME**, or **WindowsPhone8**.
- 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 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 a Worklight application

You use Worklight Studio to create the client-side of a Worklight application.

In Worklight Studio, you have two methods to create the client-side of a Worklight application:
- Use an existing Worklight project, and create your application in it, as described in "Creating an application in a Worklight project" on page 415.
- Create a Worklight project, and your application in it as its first application, as described in "Creating Worklight projects" on page 414

You can build your Worklight application for specific mobile, desktop, and web environments that you can select in Worklight Studio.
- To learn about the available environments, see "Worklight environments" on page 413
- To learn how to set up environments for your Worklight application, see "Setting up a new Worklight environment for your application" on page 438

After you create your Worklight application, you can develop its code by using different APIs:
- "JavaScript client-side API for hybrid apps" on page 418
- "Objective-C client-side API for native iOS apps" on page 418
- "Java client-side API for native Android apps" on page 418
- "Java client-side API for Java ME apps" on page 418
- "C# client-side API for Windows Phone 8 apps" on page 418

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 420 and "Developing native applications" on page 614.

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 Worklight runtime client API for mobile applications, desktop, and web to develop your applications.

For more information, see "JavaScript client-side API" on page 1002.

Objective-C client-side API for native iOS apps

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

For more information, see "Objective-C client-side API for iOS apps" on page 1005.

Java client-side API for native Android apps

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

For more information, see "Java client-side API for Android apps" on page 1006.

Java client-side API for Java ME apps

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

For more information, see "Java client-side API for Java Platform, Micro Edition (Java ME) apps" on page 1006.

C# client-side API for Windows Phone 8 apps

IBM Worklight Foundation provides the Worklight C# client-side API that you can use to develop native Windows Phone 8 applications.

For more information, see "C# client-side API for Windows Phone 8 apps" on page 1006.

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 Worklight project hierarchy:
- Your own source code files and some source code files that are provided in the 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 Foundation (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 (*)
│   │       │   │   +---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 NameIpad_Prefix.pch
│   │       │   │   │   Project Name Application NameIpad-Info.plist
│   │       │   │   │   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 a Worklight application" on page 421.

Developing hybrid and web applications

Develop hybrid and web applications as detailed here.

Anatomy of a Worklight project

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

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

Table 88. A Worklight project has the following structure:

`<project-name>`		Root project folder
	`adapters`	Source code for all adapters belonging to the project
	`apps`	Source code for all applications belonging to the project
	`bin`	Artifacts resulting from building adapters, apps, and server-side configuration and libraries
	`components`	Source code for all shell components belonging to the project
	`externalServerLibraries`	Libraries to be placed in external service servers and used for access token validation
	`www`	Source code of the Dojo JavaScript framework, if installed as part of Worklight Studio
	`server`	

Table 88. A 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 482.

Anatomy of a Worklight application

This collection of topics describes the files within a Worklight application

With IBM Worklight Foundation, 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 89. 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		

Note: If you select **Use Worklight Application Framework (beta)** when you create you application, an `application.json` file, a `view.html` file, and a `nls` folder are added to the `common` folder of your app. For more information about those resources, specific to IBM Worklight Application Framework (Beta code), see "Creating hybrid applications with IBM Worklight Application Framework" on page 559.

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 Foundation 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 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 90. 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 90. 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- *ldpi -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	

Table 90. Application icons (continued)

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

Table 90. Application icons (continued)

Environment	File name	Description	Location of default icon	Location of overriding icon
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 90. 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:

<application>

The <application> element is the root element of the descriptor. The </application> element is the closing tag.

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

The root element takes two mandatory attributes.

id   Contains the identifier of the application. The identifier 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 Worklight version on which the application was developed.

<displayName> and <description>

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

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

<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.

```
<author>
<name>ACME</name>
<email> info@acme.com </email>
<homepage> acme.com </homepage>
<copyright> (C) ACME 2014 </copyright>
</author>
```

<mainFile> and <thumbnailImage>

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

The <thumbnailImage> element contains the path to the thumbnail image for the application, including the image file name. The path is relative to the main application folder.

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

<smsGateway>

The <smsGateway> element defines the SMS gateway to be used for SMS push notifications. It has one mandatory attribute, **id**, which contains the identifier of the SMS gateway. The ID must match one of the gateway identifiers that are defined in the SMSConfig.xml file.

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

<tags> The <tags> element is supported by Android, iOS and Windows Phone 8 environments. During application deployment, the specified tags are created, updated, or deleted on the Worklight management database tables.

In the following example, the <tags> specifies customer categories. Tags represent topics of interest to the user and provide user with an ability to receive notifications according to the chosen interest. This feature enables ability for sending and receiving messages by tags. A message is targetted to only the devices subscribed for a tag.

```
<tags>
  <tag>
    <name>Silver</name>
    <description>Silver customers</description>
  </tag>
  <tag>
    <name>Gold</name>
    <description>Gold customers</description>
  </tag>
</tags>
```

environment **<version>**

Each environment on which the application can run must be declared with a dedicated XML element: <iphone>, or <ipad>. Each such element has one mandatory attribute: **version** (except for web apps). The value of the **version** attribute is a string of the form x.y, where *x* and *y* are digits (0-9).

- For mobile apps, the version is shown to users who download the app from the app store or market.
- For desktop apps, the version determines whether Worklight Server automatically downloads a new version of the app to the user's desktop.

```
<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" />
```

bundleId

In the <iphone> and <ipad> elements, you must provide the bundle ID of the application in the **bundleId** attribute. Each time the 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 because it is overridden by the builder with the value that you indicate in this attribute.

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

- 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 therefore be encrypted.

The app user can use the Worklight settings page to change the address of the Worklight Server with which the app communicates. By default, the settings page is disabled. To enable it for the app, change the **include** attribute of <worklightSettings> element to true. When the settings page is enabled, that page is accessible by using the settings app on the iOS 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 the Enabling the GCM Service page on the Android website for developers.

  The app user can use the Worklight settings page to change the address of the Worklight Server with which the app communicates. By default, the settings page is disabled. To enable it for the app, change the **include** attribute of the <worklightSettings> element to true. When the page is included in the app, for Android, a menu item is automatically appended to the options menu of the app. Users can tap this menu item to reach the page.

For more information, see the "The <security> element" on page 435.

sharedUserId

The **sharedUserId** attribute is optional. Set it only when device provisioning is activated on the application by the <authentication> element. When the **sharedUserId** attribute is set, multiple applications with the same value for this attribute can access the same keystore item on the device. The applications can thus use the same secure device ID as the one assigned to the device by the Worklight app.

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

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

<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 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, the app 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 846.

- 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: You cannot add thee <allowedDomainsForRemoteImages> subelement to the application descriptor by using the Design editor. You must use the Source editor instead.

```
<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>
```

<windows8>

The <windows8> element contains the following subelements:

- Use the <certificate> subelement to sign the Windows 8 application before you publish it. For more information, see "Signing Windows 8 apps" on page 598.

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

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

<mobileDeviceSSO>

When the <mobileDeviceSSOelement> element is specified, device single sign-on (SSO) is enabled for the application. Therefore, when a session requires authentication in a realm and a session is already active from the same device as authenticated in that realm, the authentication details from the existing session are copied to the new session. The user does not have to re-authenticate when starting the new session.

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

<air>

The optional <air> element has the following structure:

- The **showOnTaskbar** attribute determines behavior of the AIR application on the taskbar. For more information, see "Specifying the application taskbar for Adobe AIR applications" on page 596.

- Use the <certificate> element to sign the AIR application before you publish it. For more information, see "Signing Adobe AIR applications" on page 597.

- Use the <height> element to determine the height of the application on desktop environments.

- Use the <width> element to set the width of the application on desktop environments.

```
<air version="1.0" showOnTaskbar="always">
<certificate password="password" PFXFilePath="path-to-pfx"/>
<height>410</height>
<width>264</width></air>
```

<popup>

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

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

The <security> element

The <security> element is a subelement of 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:

<encryptWebResources>

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

<testWebResourcesChecksum>

The <testWebResourcesChecksum> element 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 the checksum with a value that was stored when the application 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.

<publicSigningKey>

The <publicSigningKey> element 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 462.

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

The <features> element

Since IBM Worklight V6.0.0, you can control which features are included in your application. This capability gives you a finer degree of control over the size of your application, and therefore over its capacity 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 later you add JSONStore features and want to include these resources in the application build, you can edit the <features> element. You can do so by using the Application Descriptor Editor or an XML editor, as shown in the following example:

```
<application xmlns="http://www.worklight.com/application-descriptor" id="MyProj" platformVersion="6.0
    ...
    <features>
        <JSONStore/>
    </features>
</application>
```

If you do not include JSONStore in the build but use it in your code, an error is
raised when you run the app, and you can add it to the `<features>` element with a
QuickFix.

If, during the testing phase, you find that your application does not use the
JSONStore resources, you can reduce the size of your Android app by removing
the `JSONStore` argument from the `<features>` element. When you add or remove a
feature, build the application again for the change to take effect.

Note: The JSONStore resources are still included in your iOS application builds.

For more information about the `<features>` element, see Including and excluding
application features.

The `<cacheManifest>` element

A new element now exists in the `application-descriptor.xml` named
`<cacheManifest>`. By 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 unused files, when included in the Cache Manifest file,
increase the startup time for these applications. By editing this file, you can remove
these unnecessary resources and speed up your application.

The `cacheManifest` element accepts three values, as shown in the following table.

Table 91. `cacheManifest` properties

Property	Description
no-use	In this mode (which is the default), the cache manifest is not included in the application HTML files. This setting means that there is no cache manifest and that decisions about which resources are cached are up to the browser.
generated	In this mode, the Worklight Studio builder generates a default cache manifest and includes it in the application 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.

Table 91. `cacheManifest` properties (continued)

Property	Description
user	In this mode, the Worklight Studio builder does not generate the cache manifest, but it does include it in the application 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> element with the DDE editor:
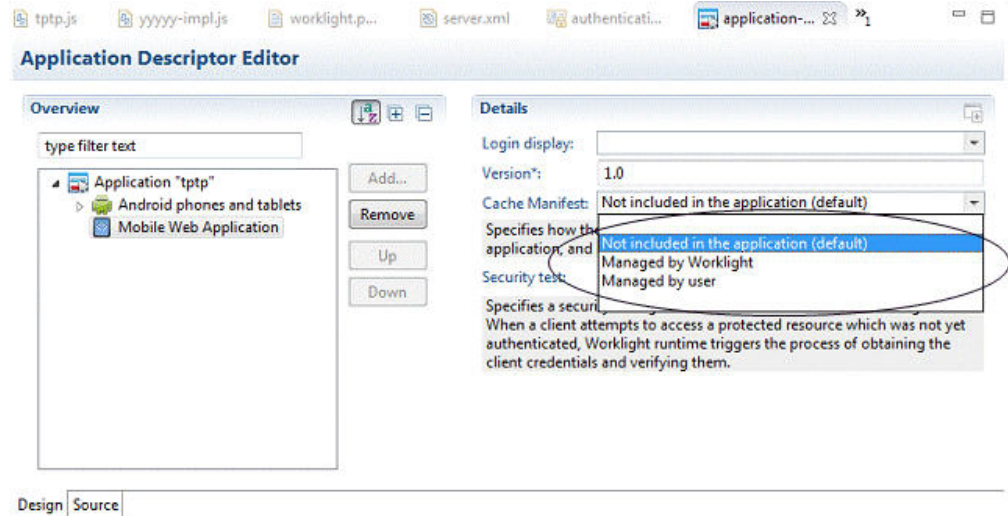
In Design view, each option is given a description:

- **Not Included in the application (default)**: **no-use** mode
- **Managed by Worklight**: **generated** mode
- **Managed by user**: **user** mode

You can also edit the value in the <cacheManifest> element of the application-descriptor.xml file itself, as shown in the following code sample:

```
<application xmlns="http://www.worklight.com/application-descriptor" id="MyProj" platformVersion="
    ...
   <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 718.

The <languagePreferences> element

The<languagePreferences> element contains a comma-separated list of languages to enforce Worklight to display system messages. For more information, see "Enforce language preference for Worklight messages" on page 955.

Deprecated elements

The following table lists deprecated and removed elements:

Table 92. Deprecated and removed elements in application descriptors

Element	Worklight version

`<provisioning>`
 	
`<viralDistribution>`
 	
`<adapters>`
 	
`<mobile>`
	Deprecated as of V4.1.3

`<worklightRootURL`	Deprecated as of V5.0
`<usage>`	Deprecated as of V5.0.0.3
`<worklightServerRootURL>`	This element was a replacement for `<worklightRootURL>` and was removed in IBM Worklight V6.0.0:

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 Worklight Server. For more information about authentication, see the module *Authentication concepts*, and the following modules under category 9, *Authentication and security*, in Chapter 4, "Tutorials and samples," on page 35.

Setting up a new Worklight environment for your application

With Worklight Studio, you can build applications for different mobile, desktop, or web environments within your 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.
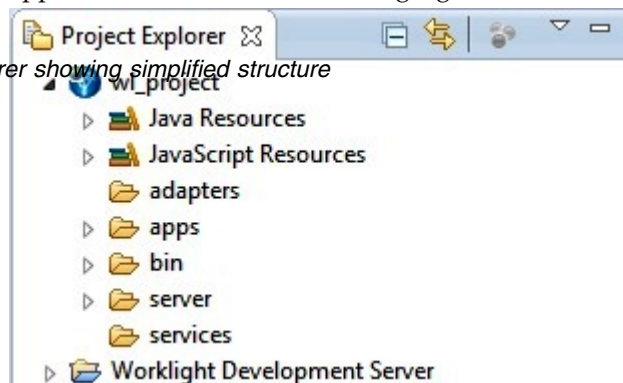
Figure 27. Project Explorer showing simplified structure

About this task

With Worklight Studio, you can add environments to your 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 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 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 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 Worklight project. To learn how to create a project, see "Creating Worklight projects" on page 414. To learn how to create an application, see "Creating an application in a Worklight project" on page 415

   You can see your new application within your Worklight project in the Project Explorer.
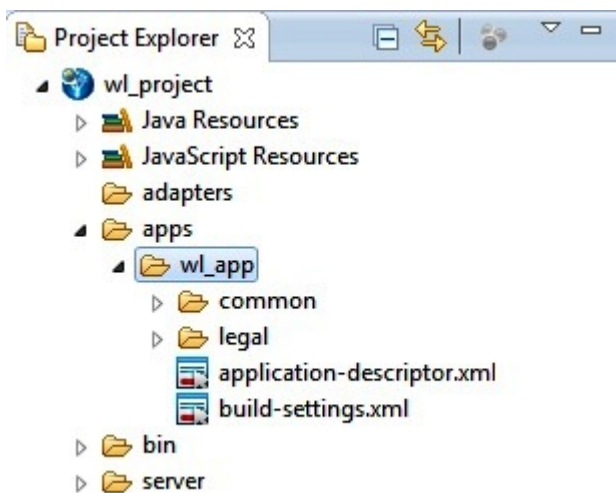
Figure 28. Your Worklight application folder

2. From the menu on top of the screen, click **File** > **New** > **Worklight Environment**.

A window opens where you can select the environment that you want to add.
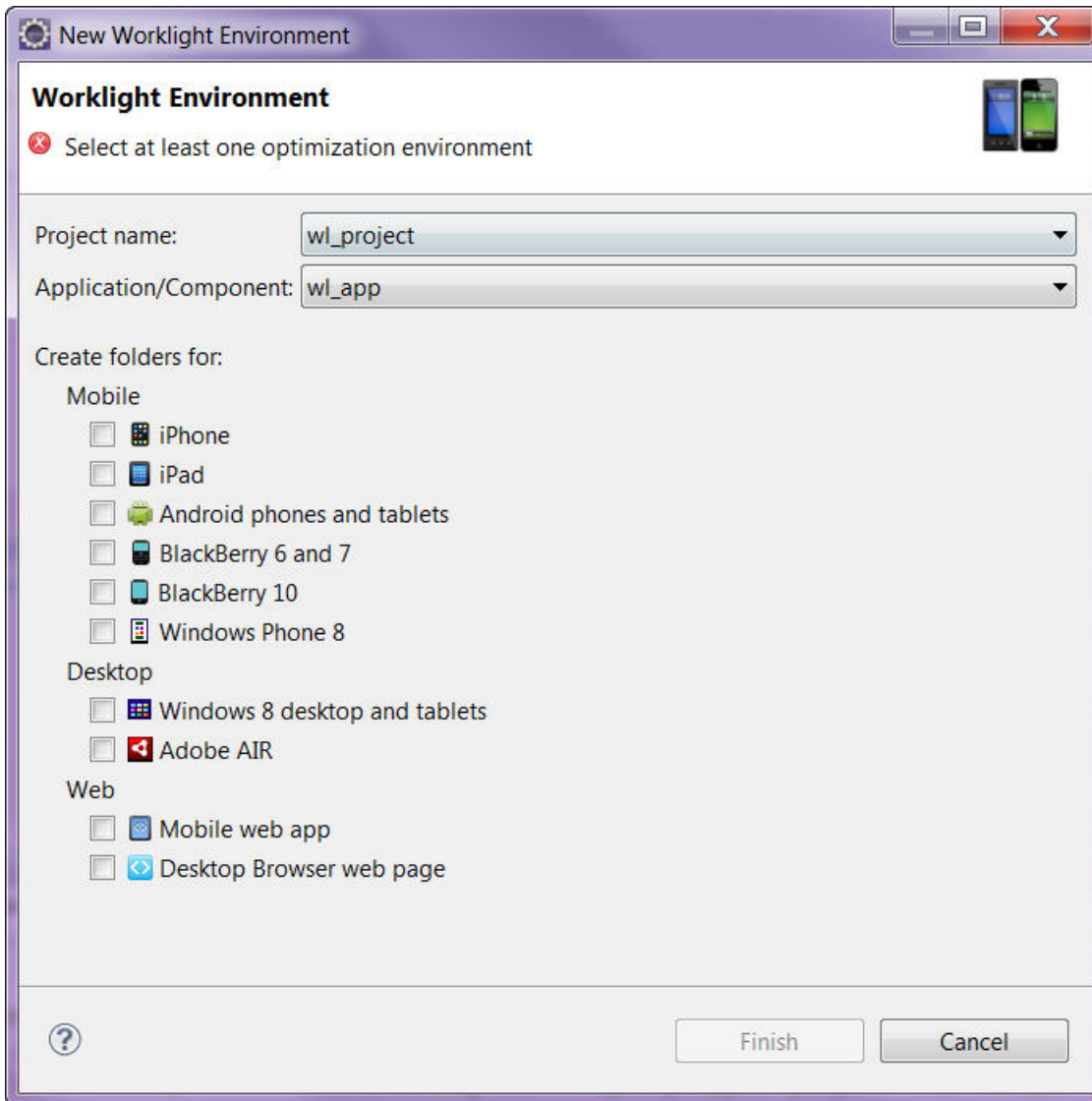
Figure 29. New Worklight Environment window

3. In the **Project name** list, select your Worklight project.
4. In the **Application/Component** list, select your Worklight application.
5. Select the environments that you want to add.
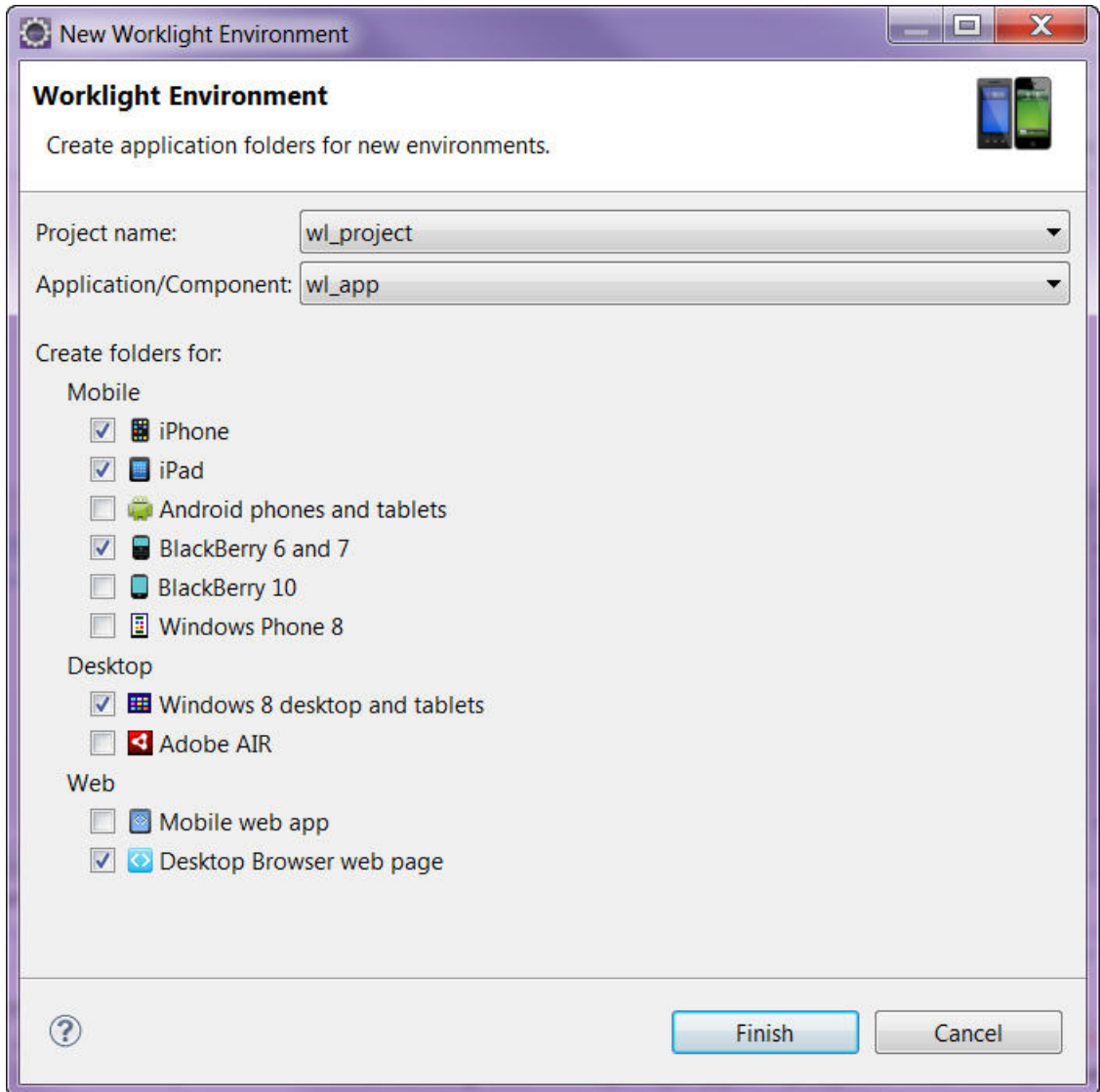
Figure 30. Selecting the mobile, desktop, and web environments that you want to add to your application

You can see the folders corresponding to the environments you added in your application folder.
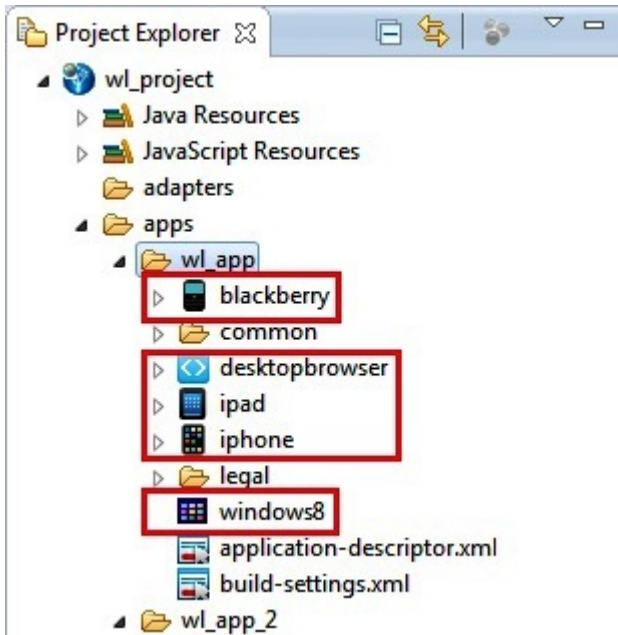
Figure 31. Worklight application folder that contains folders for the environments you selected

The Worklight Development Server and the Worklight Console

Learn about the Worklight Development Server, how it is viewed in the Worklight Console, and how to access it for Java remote debugging.

In 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.
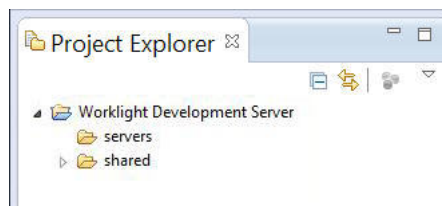
Viewing the Worklight Development Server in the Worklight Console

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.

You can 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 445.

Note: The Worklight Development Server is secured with the credentials of a user named `admin` and the password `admin`. The user name and password cannot be changed in the Worklight Development Server. When you open the Worklight Console through Worklight Studio, it is not necessary to enter the user name and password. If you keep the window of the Worklight Console open a very long time, you might receive a request to reconfirm the password.

Creating a URL to access the Worklight Console directly

In previous releases of Worklight Studio, the URL of the Worklight Console used to view the development test server in a browser had the following format:

http://localhost:*<port>*/console

Since IBM Worklight Foundation V6.2.0, in Worklight Studio, the URL of the Worklight Console has the following format:

http://localhost:*<port>*/worklightconsole

The default *<port>* after the installation of Worklight Studio is 10080. Therefore, the URL of the Worklight Console becomes:

http://localhost:10080/worklightconsole

The Worklight Console can be used to manage several projects. If only one project is deployed, this project is included directly in the URL of the console. If several projects are deployed, you can select a project from a page. The format of the URL of a project selected in this way is:

http://localhost:*<port>*/worklightconsole/index.html#*<projectname>*,catalog

The URL of the Worklight Console for a project named "myProject" becomes:

http://localhost:10080/worklightconsole/index.html#myProject,catalog

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 by 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 the Worklight Console of these servers 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:

This default port can be changed by editing the `jvm.options` file in the Worklight Studio Servers view:

Removing a project from Worklight Console

You can undeploy a Worklight project from the development test server in the **Servers** view in Eclipse.

About this task

Since the version of Worklight Studio available in IBM Worklight Foundation V6.2.0, you can use Worklight Console to administer several Worklight projects. When a new project is deployed, it is added to the development test server. If the project is no longer required, you must undeploy it manually from the development test server.

Procedure

1. To undeploy projects, open the **Servers** view in Eclipse.
2. Select **Worklight Development Server**.
3. Right-click to display the menu and select **Add and remove**.

4. Remove any projects that are no longer required. Worklight Console automatically detects that you have removed the projects.

   Note: When you remove a project from the Eclipse **Servers** view, you might see a `java.lang.NullPointerException` error message. To know how to resolve the problem, see "Troubleshooting NullPointerException error."

Troubleshooting NullPointerException error

When you remove a project from the Eclipse **Servers** view, you might see a `java.lang.NullPointerException` error message.

About this task

When you remove a project, a dialog box displays the following message: `Could not publish to the server. java.lang.NullPointerException.`

In addition, when you look at the Worklight Studio Error Log, you might find the following stack trace:

```
java.lang.NullPointerException
at com.ibm.ws.st.core.internal.WebSphereServerBehaviour.publishModules(WebSphereServerBehaviour.ja
at org.eclipse.wst.server.core.model.ServerBehaviourDelegate.publish(ServerBehaviourDelegate.java:
at org.eclipse.wst.server.core.model.ServerBehaviourDelegate.publish(ServerBehaviourDelegate.java:
at org.eclipse.wst.server.core.internal.Server.publishImpl(Server.java:3153)
at org.eclipse.wst.server.core.internal.Server$PublishJob.run(Server.java:345)
at org.eclipse.core.internal.jobs.Worker.run(Worker.java:53)
```

To resolve the problem, complete the following steps:

Procedure

1. Open the Eclipse **Servers** View.
2. Double-click **Worklight Development Server**.
3. In the **Liberty Profile Settings** section, click **Open server configuration**. The `server.xml` file opens.
4. In **Configuration Structure**, expand the **Server Configuration** menu, and click **Feature Manager**.
5. In the **Feature Manager** section on the right, click **Add**.
6. In the Add Features wizard, select **localConnector-1.0** and click **OK**.
7. Save the `server.xml` file.
8. Restart the Worklight Development Server.

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 442, 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 also work with additional *external* Worklight Servers, for example, an instance of Liberty profile that is 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 can support multiple Worklight projects, there are no longer **Start Server** and **Stop Server** menu options that are associated directly with 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/worklightconsole`.

There are two consoles now. The first is the **Worklight Console**, which contains the builder and plug-in 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 1392.

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 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.
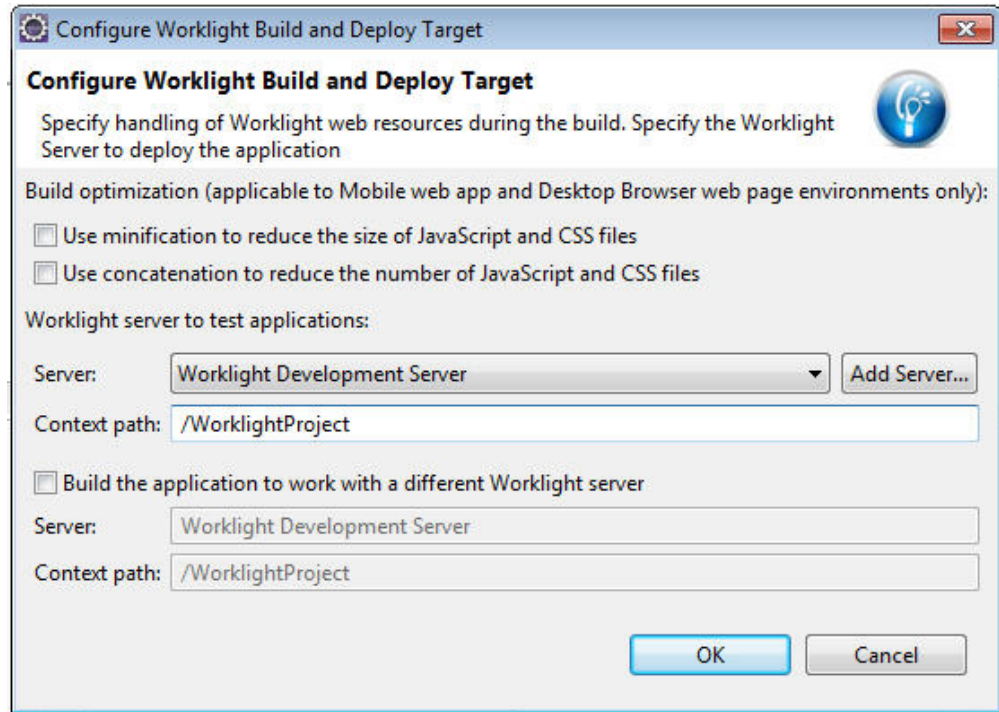
Every change that is made to the project source that is related to the project WAR file (under the `/server` folder) is automatically built and deployed to the current target server. The database connector JAR files and Worklight JAR file are also automatically deployed to this target server when you deploy the WAR file. That means that the project WAR file (not applications or adapters) is always updated on the target server. Every time that the project WAR file 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:

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 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 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 that you want to select is not visible in the **Server** dropdown list, you can add a new Worklight Server by using the following procedure. In this example, the user creates 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:
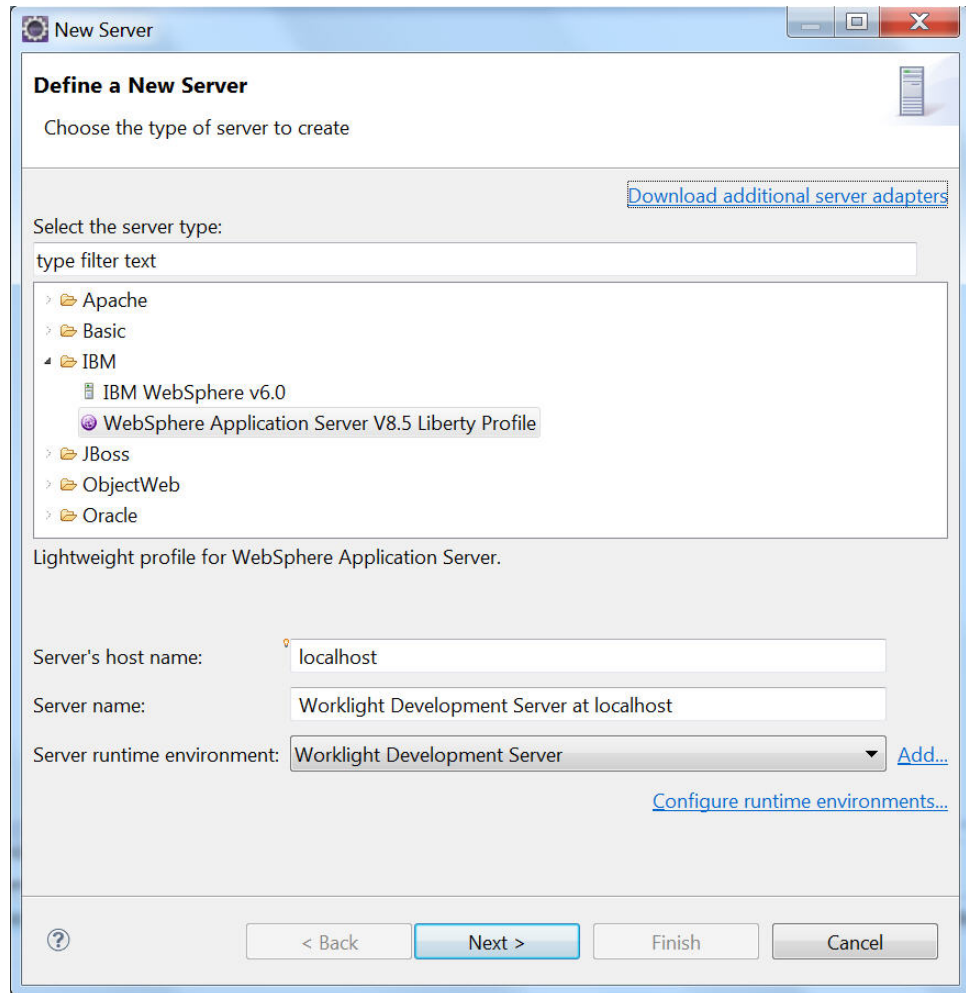
2. Select the server type that you want (in this example, **WebSphere Application Server V8.5 Liberty Profile**), and click **Add** (highlighted in the following screen capture):
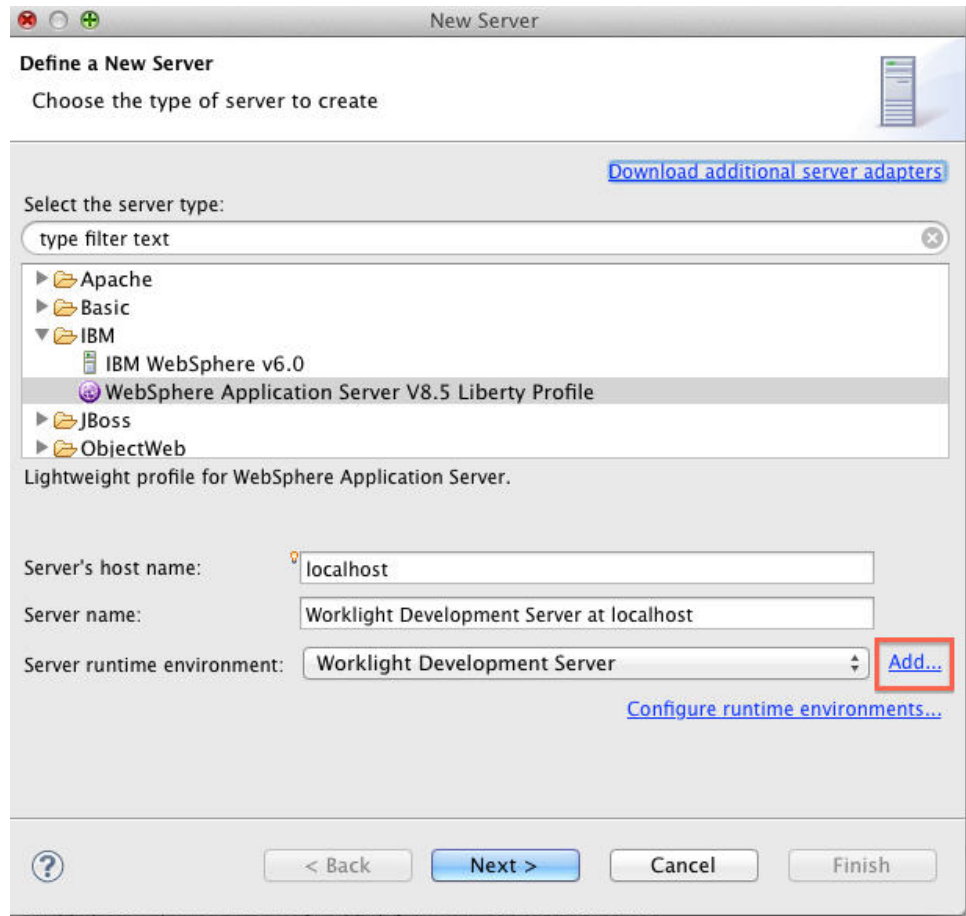
3. On the resulting screen, set **Path** to point to the directory containing the new external Liberty profile server.
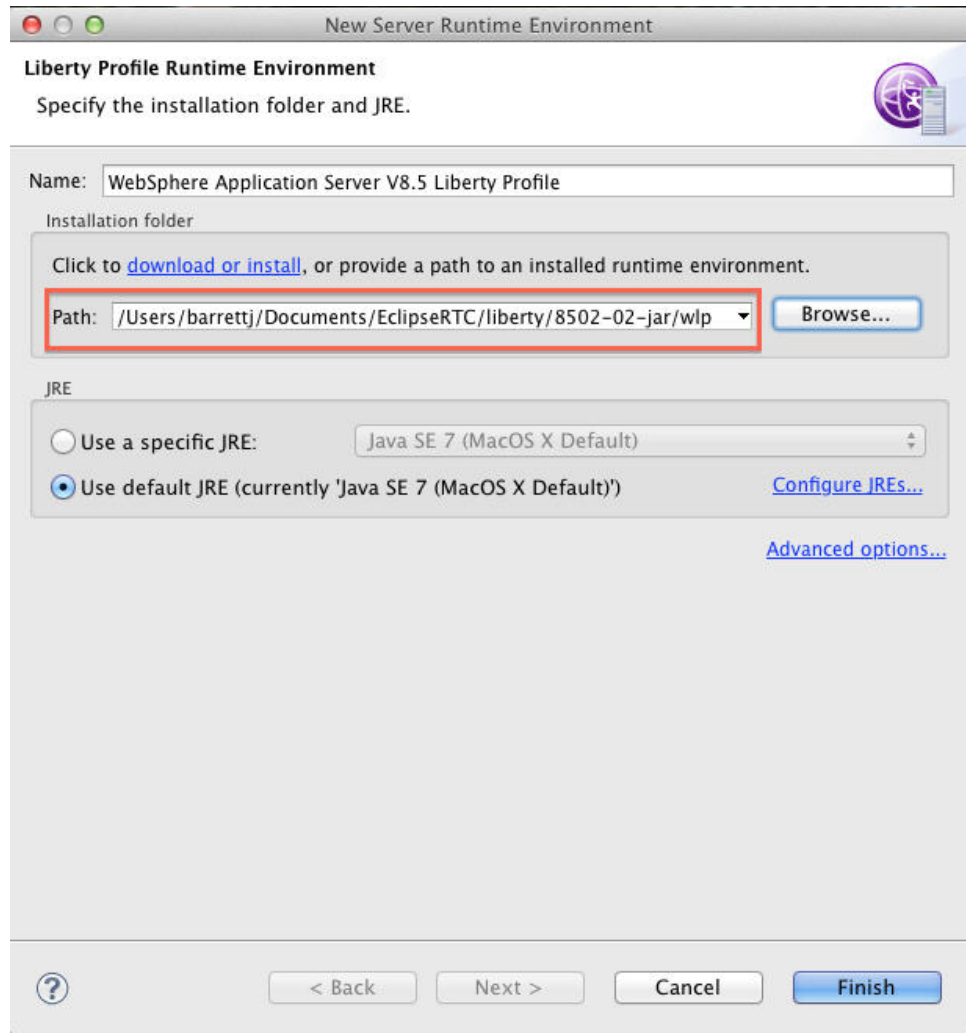
4. After you add the new server, it displays under Server runtime environment.
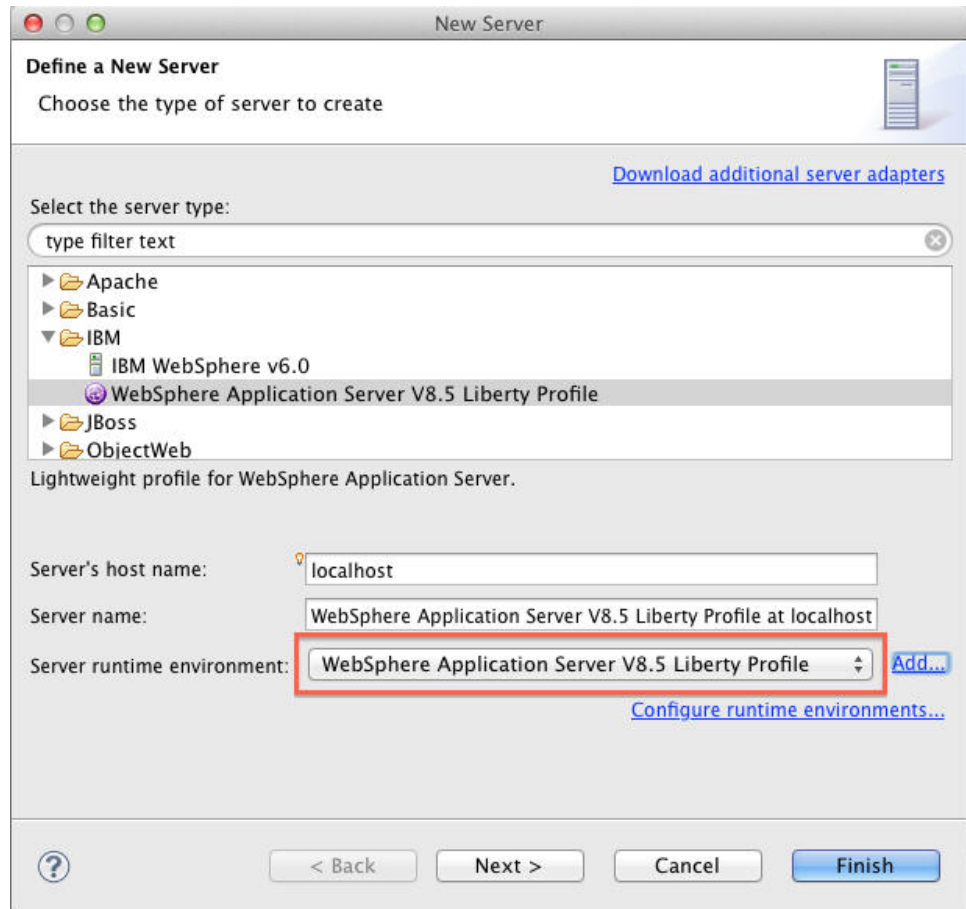
5. The new external server now is displayed in the **Server** field of the Configure Worklight Build and Deploy Target window. If you select the new server on the Configure Worklight Build and Deploy Target window, it becomes the default target test server, and all builds, deployments, and updates of 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:
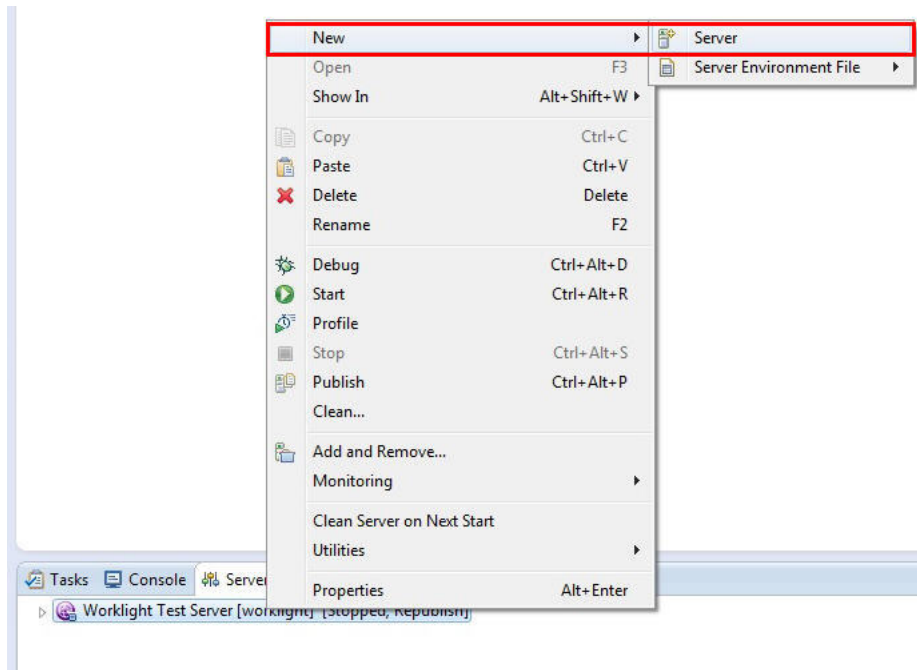
In the New Server window, select the type of server you want to add and click **Next**. Continue with the remaining screens of the New Server wizard to define your new server.

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.
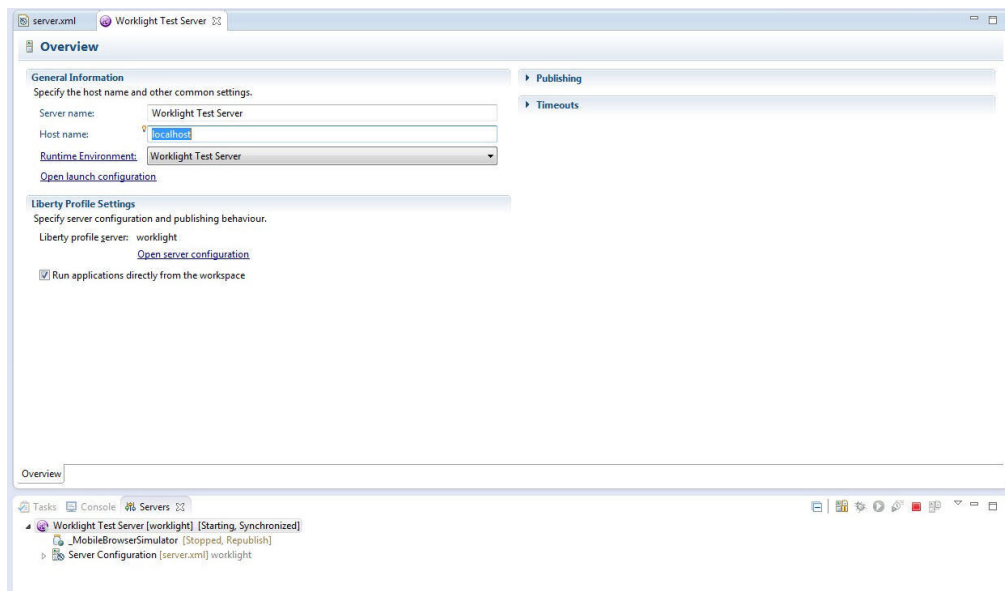
When you connect a project in Worklight Studio to an existing Liberty profile server (not the Worklight Development Server), you must check one thing before you attempt to build and deploy Worklight applications:

- In the `server.xml` file of the target server, 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 in the Liberty `server.xml` file and is related to the Liberty configuration causes a server restart.

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 hybrid applications

Use IBM Worklight Foundation to create hybrid applications.

With IBM Worklight Foundation, you can perform the following tasks:
- Add Worklight capabilities into existing native applications
- Show your splash screen as soon as the application starts
- Generally control the application startup flow and control the business logic of your apps, by running native code in your apps before they start the web view

  By default, hybrid applications start the web view immediately. However, you can use native code in the startup process and start the web view later.
- Call a Worklight hybrid web view from native code and call a native page from a web view
- Send actions and data objects between JavaScript code and native code.

  The actions are received by action receivers. Actions that cannot be delivered immediately are queued by the Worklight framework and delivered after a suitable action receiver is registered.

Developing hybrid applications for iOS
Develop hybrid applications for iOS as detailed here.

Note: The Worklight iOS SDK supports ARC (Automatic Reference Counting).The default Worklight application that is generated for the iPhone/iPad environment also supports ARC. Refer to the Apple documentation for more details on ARC.

Understanding the default startup process in iOS-based hybrid applications:

By default, the Worklight framework is initialized to display a web view in the iOS-based hybrid application. The default startup process for iOS-based hybrid applications is demonstrated here.

`initializeWebFrameworkWithDelegate` is called to start the initialization process. After the initialization process is complete, `wlInitWebFrameworkDidCompleteWithResult` is called. In case of success,

`wlInitWebFrameworkDidCompleteWithResult` creates the Cordova `ViewController` and loads the main HTML file (through `mainHtmlFilePath`) into the Cordova WebView. In case of failure, an error dialogue is displayed.

The appName.h file

```
@interface MyAppDelegate : WLAppDelegate <WLInitWebFrameworkDelegate> {

}

@end
```

Note: WLCordovaAppDelegate, CDVMainViewController, and didFinishWLNativeInit APIs are deprecated since V6.2.

The appName.m file

```
@interface Compatibility50ViewController : UIViewController
@end

@implementation Compatibility50ViewController
/**
 In iOS 5 and earlier, the UIViewController class displays views in portrait mode only. To support ac
 */
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {
    return YES;
}
@end

@implementation MyAppDelegate

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launch
{
    BOOL result = [super application:application didFinishLaunchingWithOptions:launchOptions];

    // A root view controller must be created in application:didFinishLaunchingWithOptions:
    self.window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    UIViewController* rootViewController = [[Compatibility50ViewController alloc] init];

    [self.window setRootViewController:rootViewController];
    [self.window makeKeyAndVisible];

    [[WL sharedInstance] showSplashScreen];

    [[WL sharedInstance] initializeWebFrameworkWithDelegate:self];

    return result;
}

// This method is called after the WL web framework initialization is complete and web resources are
-(void)wlInitWebFrameworkDidCompleteWithResult:(WLWebFrameworkInitResult *)result
{
    if ([result statusCode] == WLWebFrameworkInitResultSuccess) {
        [self wlInitDidCompleteSuccessfully];
    } else {
        [self wlInitDidFailWithResult:result];
    }
}

-(void)wlInitDidCompleteSuccessfully
{
    UIViewController* rootViewController = self.window.rootViewController;

    // Create a Cordova View Controller
    CDVViewController* cordovaViewController = [[CDVViewController alloc] init] ;
```

```
    cordovaViewController.startPage = [[WL sharedInstance] mainHtmlFilePath];

    // Adjust the Cordova view controller view frame to match its parent view bounds
    cordovaViewController.view.frame = rootViewController.view.bounds;

    // Display the Cordova view
    [rootViewController addChildViewController:cordovaViewController];
    [rootViewController.view addSubview:cordovaViewController.view];
}

-(void)wlInitDidFailWithResult:(WLWebFrameworkInitResult *)result
{
    UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"ERROR"
                                                        message:[result message]
                                                        delegate:self
                                                        cancelButtonTitle:@"OK"
                                                        otherButtonTitles:nil];

    [alertView show];
}
```

Implementing a custom startup process for iOS-based Hybrid applications:

You can implement a custom startup process to first display a native page in iOS-based hybrid applications.

About this task

The following application demonstrates the flexibility of the Worklight architecture: The application starts from a native page that runs custom native code. The Worklight and Cordova frameworks are used only later in the flow :

- The user clicks a button to initialize the Worklight framework. A second button is enabled only when initialization completes successfully,
- The second button loads the Cordova web view with the main HTML file of the application.

Procedure

1. Create a hybrid app, which creates the .h file that extends WLAppDelegate. For example:

```
@interface MyAppDelegate : WLAppDelegate
{   }
@end
```

   Note: WLCordovaAppDelegate, CDVMainViewController, and didFinishWLNativeInit APIs are deprecated since V6.2.0.

2. Create a custom native view.

   Note: The Worklight framework is not used at this point.
   For example:

```
@implementation MyAppDelegate
    // Create a custom view before initializing the Worklight framework
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *
{
    BOOL res = [super application:application didFinishLaunchingWithOptions:launchOptions];

    // code for building a window

    // Build custom native UI
    MyCustomViewController* customViewController = [[MyCustomViewController alloc]init];

    // Set root view controller
```

```
                                    self.window.rootViewController = customViewController;

                                    // Show window
                                    [self.window makeKeyAndVisible];

                                    return res;
                                }
                                @end
```

3. Create a custom view controller. The view controller performs some custom
   business logic, and contains two buttons

   • one to trigger the Worklight initialization

   • one to load the Cordova Webview

   A custom delegate is used to handle the UI actions and to respond to the
   initialization completion For example:

```
@interface MyCustomViewController : UIViewController
{    }
@property (retain, nonatomic) MyWLDelegate* wlDelegate;
@end

@implementation MyCustomViewController
//
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
        init wlDelegate
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    //Run some business logic here
}
    //Run some more business logic

// Start Worklight initialization
- (IBAction)onInitWLClicked:(id)sender {
    [[WL sharedInstance] initializeWebFrameworkWithDelegate:self.wlDelegate];}

// Load the Cordova web view
- (IBAction)onShowWebViewClicked:(id)sender {
    [self.wlDelegate showCordovaWebView];
}}
@end
```

4. Create the custom delegate that implements WLInitWebFrameworkDelegate
   and some custom logic. For example:

```
@interface MyWLDelegate : NSObject <WLInitWebFrameworkDelegate>
//
@end

@implementation MyWLDelegate

// Implement WLInitWebFrameworkDelegate protocol
-(void)wlInitWebFrameworkDidCompleteWithResult:(WLWebFrameworkInitResult *)result
{
    if ([result statusCode] == WLWebFrameworkInitResultSuccess) {
        [self wlInitDidCompleteSuccessfully];
    } else {
        [self wlInitDidFailWithResult:result];
    }
}

-(void)wlInitDidCompleteSuccessfully{
    // Enable the button on the viewController
```

```
    }

    // load and show the Cordova WebView
    -(void)showCordovaWebView
    {
        // build the cordova view controller
        CDVViewController* cordovaViewController = [[[CDVViewController alloc]init]];

        // Set URL of main HTML file
        cordovaViewController.startPage = [[WL sharedInstance ]mainHtmlFilePath];

        // Show the Cordova web view
        UIViewController* rootViewController = self.window.rootViewController;
        [rootViewController addChildViewController:cordovaViewController];
        [rootViewController.view addSubview:cordovaViewController.view];
    }

    // Handle initialization failure
    -(void)wlInitDidFailWithResult:(WLWebFrameworkInitResult *)result
    {
        UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:@"ERROR"  message:[result mess
        [alertView show];
        [alertView release];
    }
    @end
```

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.

Customizing iOS applications:

You can further customize your iOS application by extending a different class than
the default Worklight class.

Before you begin

These features require you to create your application in IBM Worklight Foundation
V6.2.0 and later. If you created your application in a previous version of IBM
Worklight, follow the upgrade procedures to upgrade your application to V6.2.0.

About this task

By default, the application delegate extends the WLAppDelegate class, which provides you with access to various features provided by IBM Worklight Foundation, such as push notifications and local notifications. However, if your application delegate extends a class other than WLAppDelegate, you can still access Worklight features by following these steps.

Procedure

1. Create a hybrid iOS application
2. In MyAppDelegate.h, have MyAppDelegate extend a different class than WLAppDelegate. For example,

   `MyAppDelegate : UIResponder <WLInitWebFrameworkDelegate,UIApplicationDelegate>`
3. Add the **window** property to MyAppDelegate.h, following iOS guidelines.

   `@property (nonatomic, retain) IBOutlet UIWindow* window;`
4. Add the **launchOptions** property to MyAppDelegate.h

   `@property ( nonatomic, retain) NSMutableDictionary* launchOptions;`
5. Synthesize the two properties in MyAppDelegate.m

   `@synthesize window, launchOptions;`
6. In MyAppDelegate.m, make the following changes to applicationDidFinishLaunchingWithOptions:

   a. Make necessary changes to the super call. In this example, remove the line

   `BOOL result = [super application :application didFinishLaunchingWithOptions:launchOptions]`

   and return **YES** instead of result.

   b. Add the following line to keep the launchOptions. For example,

   `self . launchOptions = [ NSMutableDictionary dictionaryWithDictionary :launchOptions];`

Results

You can now run your application and access most of the Worklight features. Some features, such as custom URL schemas, local notifications, and push notifications require extra steps.

What to do next

To enable the use of custom URL schemas, add the following method to your application delegate (MyAppDelegate), and follow the Cordova documentation on **handleOpenURL**

```
- (BOOL)application:( UIApplication*)application handleOpenURL:(NSURL *)url
{
    [[ NSNotificationCenter defaultCenter ] postNotificationName: WLapplicationHandleOpenURL object
    return YES;
}
```

To use Local Notifications, add the following method to your application delegate (MyAppDelegate), and follow the Cordova documentation on sending local notifications to Cordova plug-ins.

```
- (void)application:( UIApplication*)application
didReceiveLocalNotification:( UILocalNotification*)notification
{
    [[NSNotificationCenter defaultCenter ] postNotificationName: WLapplicationDidReceiveLocalNotifica
}
```

To enable Worklight Hybrid Push Notifications, add the following three methods
to your application delegate (MyAppDelegate), and follow the Worklight
documentation on Hybrid Push Notifications.

```
- (void)application:( UIApplication *)application didRegisterForRemoteNotificationsWithDeviceToken
{
    [[ NSNotificationCenter defaultCenter ] postNotificationName :WLapplicationDidRegisterForRemot
}
- ( void)application:(UIApplication *)application didFailToRegisterForRemoteNotificationsWithError
    [[ NSNotificationCenter defaultCenter ] postNotificationName :WLapplicationDidFailToRegisterFo
}
- ( void)application:(UIApplication *)application didReceiveRemoteNotification:( NSDictionary *)us
    [[ NSNotificationCenter defaultCenter ] postNotificationName :WLapplicationDidReceiveRemoteNo
}
```

Developing hybrid applications for Android
Develop hybrid applications for Android as detailed here.

Note: When Android runs in debuggable mode, which can be set in the
application's manifest file, 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.

Note: If you are targeting devices on API level below 14, add the following
permission to your AndroidManifest.xml file: <uses-permission
android:name="android.permission.GET_TASKS"/>. This permission is required for
the heartbeat functionality to function properly.

Understanding the Default startup process in Android-based Hybrid applications:

By default, the Worklight framework is initialized to display a web view in the
Android-based hybrid application. The default startup process for Android-based
hybrid applications is described here.

The default activity

The default activity for the Android environment extends CordovaActivity and
implements WLInitWebFrameworkListener.

For example:

```
public class AndroidWebview extends CordovaActivity implements WLInitWebFrameworkListener {

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

Note: The WLDroidGap API is deprecated since V6.2.

WL APIs

The WL APIs are called before any other IBM Worklight Foundation API to initialize the IBM Worklight Foundation framework.

For example:

```
//Create Worklight framework, show splash screen, and initialize
//the web framework, passing 'this' as the listener for the web
//initialization process

WL.createInstance(this);

WL.getInstance().showSplashScreen(this);

WL.getInstance().initializeWebFramework(getApplicationContext(), this);

// Add additional code here
}
```

Initialization status

If the initialization status is **SUCCESS**, the webpage is loaded with the WL.getMainHtmlFilePath() method. For example:

```
@Override

//Start the Cordova web view using the main HTML file path that it gets
//from Worklight framework

public void onInitWebFrameworkComplete(WLInitWebFrameworkResult result){
  if(result.getStatusCode() == WLInitWebFrameworkResult.SUCCESS) {
    super.loadUrl(WL.getInstance().getMainHtmlFilePath());
```

Initialization failure

The initialization status can be:
- FAILURE_INTERNAL
- FAILURE_UNZIP
- FAILURE_CHECKSUM
- FAILURE_NOT_ENOUGH_SPACE

For example:

```
} else {
  AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
  alertDialogBuilder.setNegativeButton(R.string.close, new OnClickListener() {
    @Override
    public void onClick(DialogInterface dialog, int which) {
      finish();
    }
  }};

  alertDialogBuilder.setTitle(R.string.error);
  alertDialogBuilder.setMessage(result.getMessage());
  alertDialogBuilder.setCancelable(false);
  alertDialogBuilder.create();
  alertDialogBuilder.show();
}
```

Implementing a custom startup process in Android-based hybrid applications:

You can display a native page when you start an Android-based hybrid application.

About this task

You can choose to initialize the Worklight framework whenever you want to display a web view.

Procedure

1. Create the main launcher activity that extends Activity and implements WLIniWebFrameworktListener. For example:

```
public class NativeCustomActivity extends Activity implements WLInitWebFrameworkListener {

 public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);

  WL.createInstance(this);
  WL.getInstance().initializeWebFramework(getApplicationContext(), this);


 public void onInitWebFrameworkComplete(WLInitWebFrameworkResult result){
  if(result.getStatusCode() == WLInitWebFrameworkResult.SUCCESS) {
   openMainNativeActivity();
  }
  else {
    handleWebFrameworkInitFailure(result);
  }
 }
}
```

   Note: The WLDroidGap API is deprecated since V6.2.
2. Create the main UIActivity that extends Activity. For example:

```
public class NativeMainActivity extends Activity {

    public void onStart(){
        super.onStart();
        // The rest of the code for regular native activity,
        // and opening a CordovaActivity when ever it's needed
        openHybridPage();
 }
```

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.

Extracting a public signing key:

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

Procedure

1. In the Eclipse project explorer, right-click the android folder for the application and then click the **Extract public signing key** menu item.
   A wizard window opens.
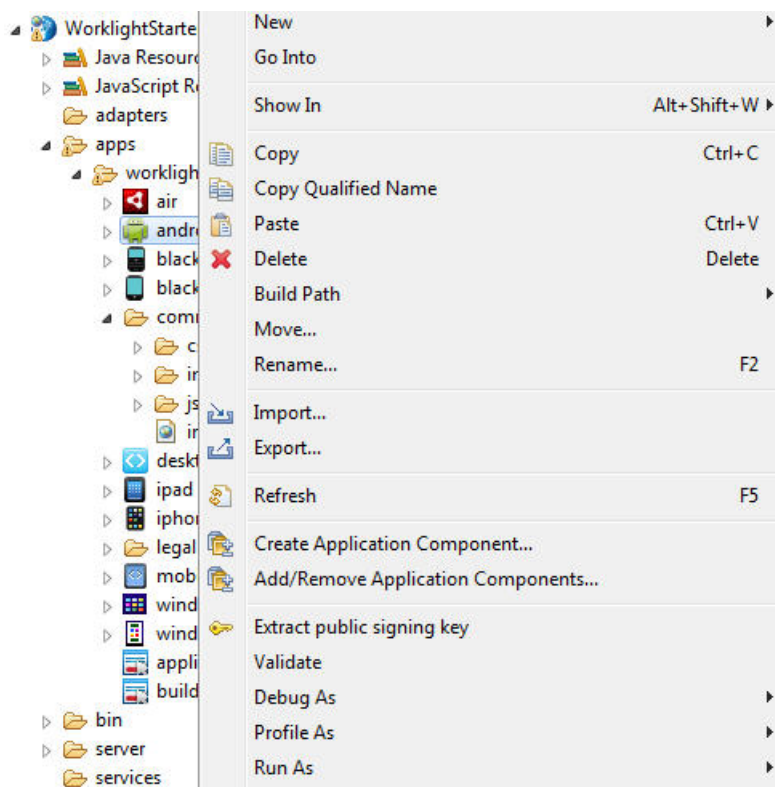
Figure 32. Extracting the public signing key
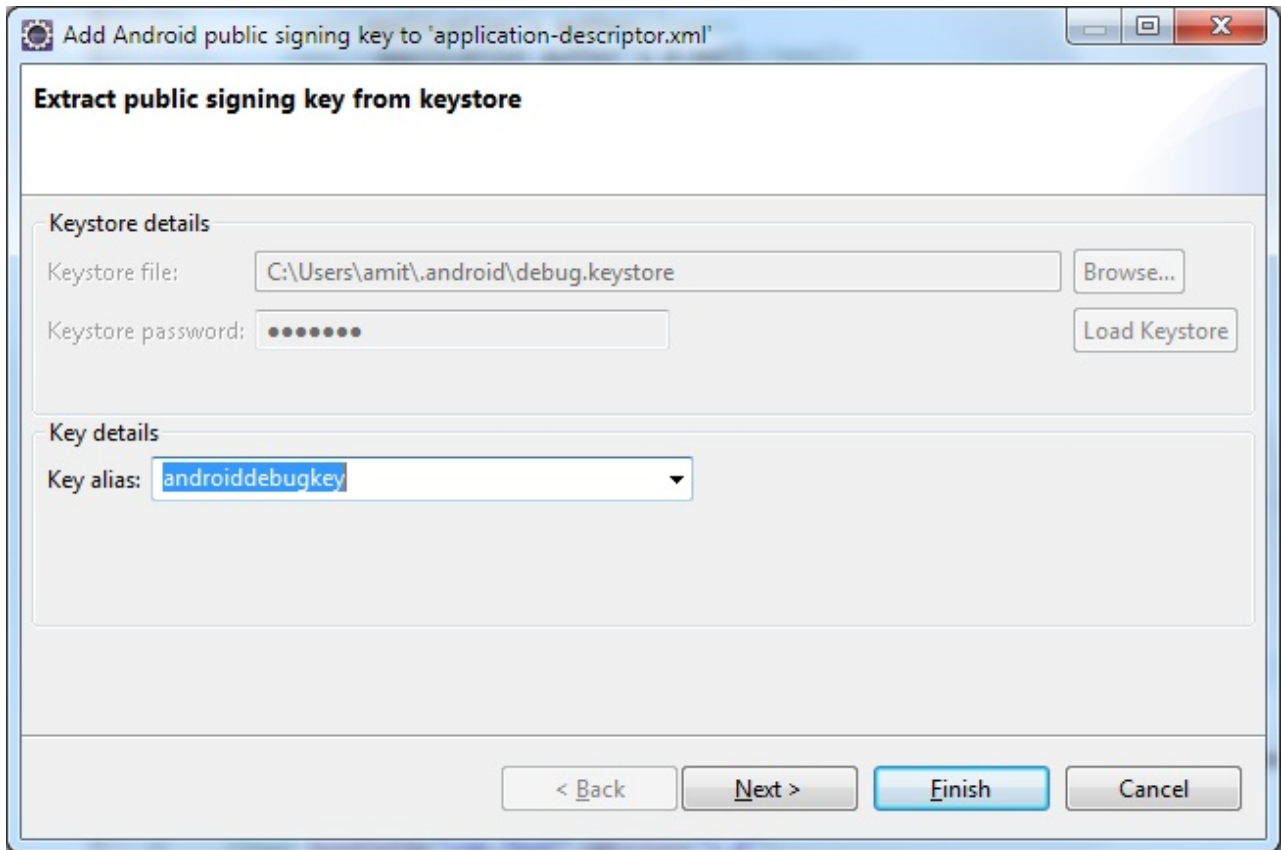
Figure 33. Adding the Android public signing key

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	`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.

Figure 34. 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="false"/>
  <security>
    <testAppAuthenticity enabled="false"/>
    <encryptWebResources enabled="false"/>
    <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
    <publicSigningKey>MIGfMA0CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCE+TiHbDxPx0HA6rARXoJWC071hLLBytTDSdNe/></publicSigningKey>
  </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 Foundation 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:configChanges=
    ```

- 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 the directory that contains 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), click **Browse**, and specify the directory where your Android Studio is installed.
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 Foundation 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 a Worklight BlackBerry 10 environment:

Follow these instructions to create a Worklight BlackBerry 10 environment.

About this task

> **Note:**
> - The instructions in the Procedure section are valid for IBM Worklight Foundation up to V6.2.0.1 with WebWorks SDK 1.x.
> - If a fix pack such as V6.2.0.2 is installed, you must complete the instructions based on the version of WebWorks SDK:
>
>   **For WebWorks SDK 1.x:**
>   – Follow the instructions in the Procedure section.
>
>   **For WebWorks SDK 2.x:**
>   – Follow the instructions of BlackBerry documentation. For more information, see Building and Testing.
> - BlackBerry OS 10 is not supported by the current version of the Application Center.

The BlackBerry 10 environment uses the latest version of Cordova. Use either Ripple or Cordova Ant scripts to create a Worklight BlackBerry 10 environment, and follow these steps to ensure that your program runs correctly.

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 via
 # 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.

> **Note:** This content is valid for IBM Worklight Foundation up to V6.2.0.1. If V6.2.0.2 fix pack is installed, you can ignore the following instructions. With V6.2.0.2 fix pack, the BlackBerry 10 environment that is created in the Worklight application has a BlackBerry 10 `native` folder with WebWorks SDK 2.x project structure.

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 Foundation do not work as-is with WebWorks 2.0. Perform the following instructions to make IBM Worklight Foundation 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.
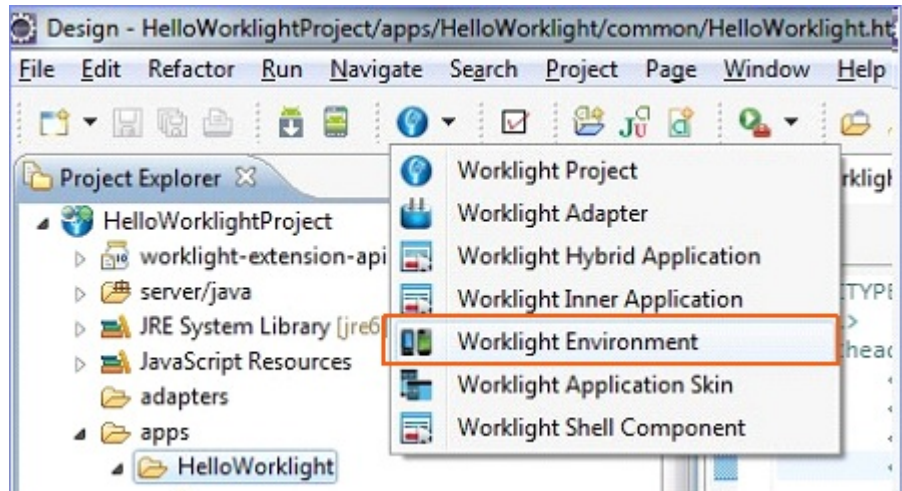
4. Select **BlackBerry 10**, and click **Finish**.
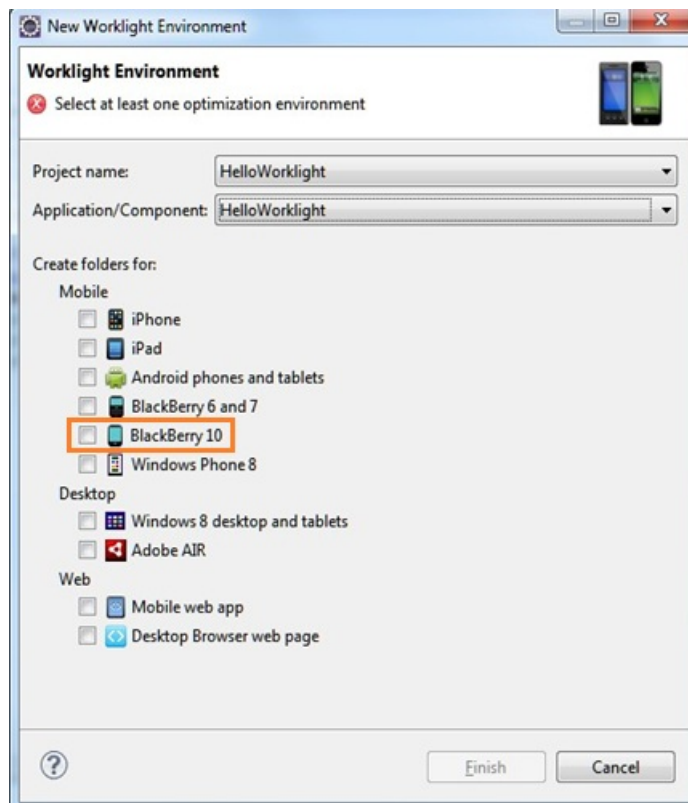
5. A `blackberry10` environment folder is automatically added. This
   environment folder includes the following subfolders:

   - `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.
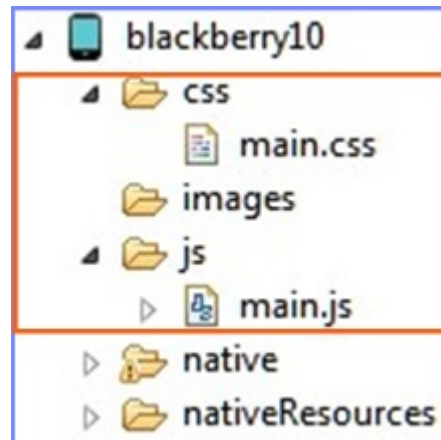
Upgrade Worklight BlackBerry 10 Project

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.
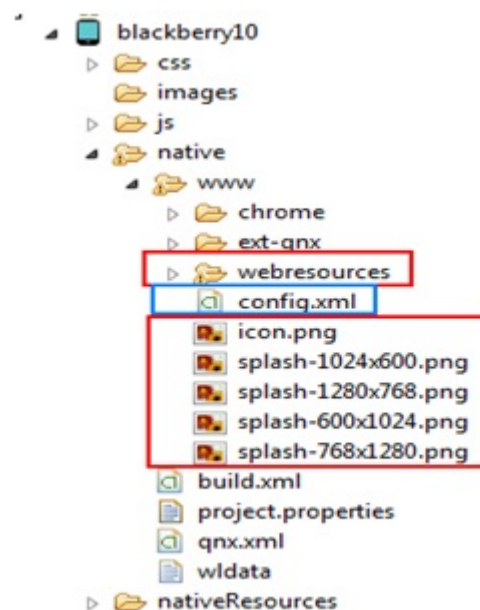
5. Paste the copied files under *project_name*/www folder of the WebWorks SDK 2.0 project.
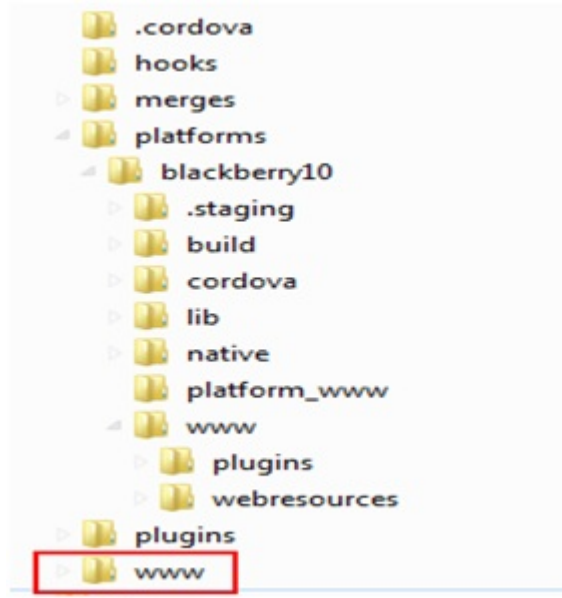
6. Copy the `config.xml` file of the Worklight BlackBerry 10 environment folder, and paste it under the root folder of WebWorks SDK 2.0 project. Replace the existing one.

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.

Migrating a BlackBerry 10 WebWorks 1.x Worklight project to WebWorks 2.x:

Follow these instructions to make a Worklight BlackBerry 10 WebWorks 1.x project works with WebWorks SDK 2.x.

Note: These instructions are valid for IBM Worklight Foundation V6.2.0.2 or future fix pack.

1. Set the `WEBWORKS_HOME` environment variable to the WebWorks SDK 2.x folder.
2. Create a Worklight project and Worklight application.
3. Add BlackBerry 10 environment to it. A BlackBerry 10 `native` folder is created with WebWorks SDK 2.x project structure.
4. Replace *new_project*/*new_app*/blackberry10/native/www folder with *old_app*/blackberry10/native/www folder of the old project. This old project has a BlackBerry 10 `native` folder with WebWorks SDK 1.0 project structure.
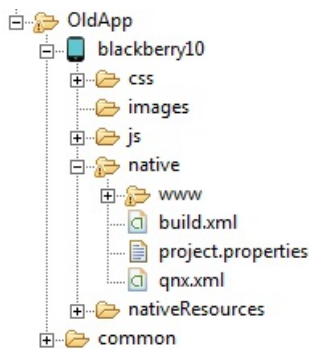
Figure 35. `native` *folder of an old project with WebWorks SDK 1.0 project structure*
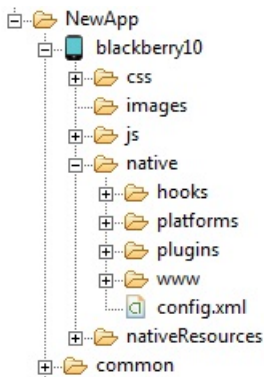
Figure 36. `native` *folder of a new project with WebWorks SDK 2.x project structure*

5. Replace *new_project*/*new_app*/blackberry10/css, *new_app*/blackberry10/images, *new_app*/blackberry10/js, and *new_app*/common folder with corresponding folders from the old project.
6. Modify *new_project*/*new_app*/blackberry10/native/www/webresources/default/index.html file with the following steps:

        a. Remove `<script>` tag that includes `webworks.js`.

        b. Change `src="worklight/cordova.js"` to `src="../../cordova.js"`. The path now points to *new_project*/*new_app*/`blackberry10/native/platforms/` `blackberry10/www/cordova.js`.

7. Remove unnecessary files and folders from *new_project*/*new_app*/`blackberry10` folder:

        a. Delete `chrome`, `ext-qnx`, and `config.xml` from `native/www` folder.

        b. Delete `Plugins`, `cordova.js`, `cordova_plugin.js` from `native/www/` `webresources/default/worklight/` folder.

8. Reapply the custom changes that you made to the `config.xml` file. Modify the *new_app*/`blackberry10/native/config.xml` file with the changes made in the `config.xml` file of the old project. For example, changes in `rim:splash`, `content`, and `rim:permission` elements.

9. If any Cordova and BlackBerry plug-ins are added to your old project, add them to the new project.

10. The new project that you created is your migrated project. You can archive the old project for future reference.

For more information, see Blackberry documentation: Upgrading from WebWorks 1.0.

Developing hybrid applications for Windows Phone

Redeploying a Windows Phone 8 app from Microsoft Visual Studio while deploying your changes to the server from Worklight Studio can trigger the direct update process.

Whenever you deploy your changes to the server from Worklight Studio and at the same attempt to redeploy a Windows Phone 8 app from Microsoft Visual Studio, the direct update dialog is displayed notifying the availability of an update. This happens because when you update your app, any data in the isolated storage is preserved.

This behavior occurs only when you have already done a direct update once. Also, note that isolated storage is a place where preferences and direct update resources are stored. For more information about isolated storage, see Isolated Storage.

Managing the splash screen

Show and hide the splash screen.

Managing the splash screen with JavaScript APIs:

You can choose to show and hide the splash screen with JavaScript.

In JavaScript code, you can use the WL.App.showSplashScreen() and WL.App.hideSplashScreen() methods to show and hide the splash screen.

By default, the Worklight JavaScript library auto-hides the splash screen when the application is launched. You can choose to disable the automatic hiding of the splash screen by setting the option **autoHideSplash** to `false` in the `initOptions.js` file and use the WL.App.hideSplashScreen() method to hide the splash screen after all of the page initialization tasks (including loading other JavaScript frameworks) are completed.

In case the application main web file is set to an external web page (the URL for the file starts with http:// or https://), the splash screen is automatically hidden when the web view finishes loading the page.

Managing the splash screen in an iOS-based hybrid application:

You can choose how to manage the splash screen for iOS during application initialization.

By default, iOS hybrid applications show a splash screen image during application initialization. This image is selected at run time from the launch images that are supplied in the application Xcode project. You can also use a custom splash screen by replacing the default set of splash images in the `native/Resources` folder of your Worklight project.

Note: iOS automatically displays the application launch image when the app is launched, and hides the image when the app is ready (after `applicationDidFinishLaunchingWithOptions`). After this image is removed, you can use the Worklight splash screen as required by the logic of your application. The documentation in this section describes the Worklight splash screen API.

The code that shows the splash screen is in the `{AppName}.m` class in the `didFinishLaunchingWithOptions` method. You can use the [[WL.sharedInstance] showSplashScreen] API to show the splash screen.

Note: You must define a root view controller before you call this API. For example:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)lau
{
  ...
  UIViewController* rootViewController = ...
  [self.window setRootViewController:rootViewController];
   ...
  [[WL sharedInstance] showSplashScreen];
  ...
}
```

By default, the Worklight JavaScript library auto-hides the splash screen when the application is launched. To have a smooth transition from the splash screen to the web view, set the option **autoHideSplash** to `false` in the `initOptions.js` file and use the WL.App.hideSplashScreen() method to hide the splash screen after all of the page initialization tasks (including loading other JavaScript frameworks) are completed.

Note: Make sure that your application initialization flow does not block the JavaScript call to hide the splash screen. For example, a problem can occur when you set the application to connect to the server on application startup, and you define form-based authentication that waits for the user to enter login credentials. In this case, the application shows a web login form behind the splash screen without a way for the user to interact with it.

Note: If you use the splash screen API offered in Cordova, do not use the Worklight splash screen APIs at the same time.

Enabling high-resolution splash images for iPhone 6 and 6 Plus devices:

Use a **UILaunchImages** dictionary to enable high-resolution splash images.

During app startup on earlier versions of iPhone, iOS looked for splash images among application resources. IBM Worklight looked for the same images and displayed them during WebView initialization. For iPhone 6 and 6 Plus, however, iOS is unable to automatically detect high-resolution images and IBM Worklight is therefore not aware of them. To enable a high-resolution splash screen image on iPhone 6 and 6 Plus, developers must create a dictionary that references the set of images in the application resources. The dictionary comprises **UILaunchImages** key-value pairs and must be located in the Info.plist file. When the reference to the set appears in the Info.plist, both iOS and IBM Worklight are able to select the splash image that is most suited to the device resolution.

The following code is a sample dictionary. To enable default Worklight splash images, copy the code as-is into the Info.plist file to enable high-resolution splash images for iPhone 6 or 6 Plus, along with low-resolution images for iPhone 3, 4, and 5.

Note: Images with lower resolutions must also appear in the dictionary. The code uses the following image names:

- Default-667h.png: image for iPhone 6
- Default-736h.png: image for iPhone 6 Plus
- Default-736h-Landscape.png: image for iPhone 6 Plus in landscape orientation

Of course, you can customize the code to use your own images.

Note: The name of the splash image must be identical in both the dictionary and the application resources.

```
<key>UILaunchImages</key>
<array>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
        <string>8.0</string>
        <key>UILaunchImageName</key>
        <string>Default</string>
        <key>UILaunchImageOrientation</key>
        <string>Portrait</string>
        <key>UILaunchImageSize</key>
        <string>{320, 480}</string>
    </dict>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
        <string>8.0</string>
        <key>UILaunchImageName</key>
        <string>Default-568h</string>
        <key>UILaunchImageOrientation</key>
        <string>Portrait</string>
        <key>UILaunchImageSize</key>
        <string>{320, 568}</string>
    </dict>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
        <string>8.0</string>
        <key>UILaunchImageName</key>
        <string>Default-667h</string>
        <key>UILaunchImageOrientation</key>
        <string>Portrait</string>
        <key>UILaunchImageSize</key>
        <string>{375, 667}</string>
    </dict>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
        <string>8.0</string>
```

```
        <key>UILaunchImageName</key>
        <string>Default-736h</string>
        <key>UILaunchImageOrientation</key>
        <string>Portrait</string>
        <key>UILaunchImageSize</key>
        <string>{414, 736}</string>
    </dict>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
        <string>8.0</string>
        <key>UILaunchImageName</key>
        <string>Default-736h-Landscape</string>
        <key>UILaunchImageOrientation</key>
        <string>Landscape</string>
        <key>UILaunchImageSize</key>
        <string>{414, 736}</string>
    </dict>
</array>
```

Managing the splash screen in an Android-based hybrid application:

You can choose how to manage the splash screen for Android during application initialization.

Java APIs for showing and hiding the splash screen

In Java, you can use the methods WL.getInstance().showSplashScreen(Activity activity) and WL.getInstance().hideSplashScreen() to show and hide the splash screen from native code.

Changing the default splash image

You can change the default splash image that is located in the res/drawable folder and is named splash.png.

Disabling the splash screen in Android

For Android, you can disable the splash screen either by:
- Editing the Native Android Appclass and removing or commenting out the WL.getInstance().showSplashScreen(this) API call
- Deleting the splash.png file in the res/drawable folder.

Cordova splash screen API

If you use the splash screen API offered in Cordova, do not use the Worklight splash screen APIs at the same time.

Showing a custom image

You can show a custom image when you use Cordova splash screen APIs by storing the image in the res/drawable folder and then either:
- Declaring it in the config.xml file.

  ```
  <preference name="SplashScreen" value="splash_for_cordova_activity" />
  <preference name="SplashScreenDelay" value="60000" />
  ```
- Adding it programmatically in the CordovaActivity class.

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

```
//Add these two lines to enable the Cordova splash screen

    this.splashscreen = R.drawable.splash_for_cordova_activity;
    this.showSplashScreen(60000);

    this.loadUrl(WL.getMainHtmlFilePath());
}
```

When the web view is ready to be shown, you can hide the splash screen by using:

`navigator.splashscreen.hide();`

Showing a custom splash screen by adding a custom activity

You can implement a custom splash screen by adding a custom activity to be used as a splash screen. Here is an example of code where you to declare an activity in `AndroidManifest.xml`.

Note: The activity must have an intent-filter with `MAIN` and `LAUNCHER` properties that are defined in order to be launched.

```
<activity
        android:name="com.MyApp.MySplashScreen"
        android:label="@string/app_name"
        android:screenOrientation="portrait"
        android:theme="@android:style/Theme.Black.NoTitleBar" >
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
```

In your activity onCreate() method, after you show the custom UI, you can use the WL.createInstance().initializeWebFramework() API to initialize Worklight. Once initialization is complete, you can hide your custom splash screen activity or move to a different CordovaActivity in order to show the application's web view.

Showing a custom loading spinner

You can enable a Cordova loading spinner by declaring it in the `custom.xml` file.

`<preference name="LoadingDialog" value="Please wait while the application loads."/>`

Note: You can also set the Cordova Activity background color to transparent instead of the default black by declaring it in the `config.xml` file.

`<preference name="BackgroundColor" value="0"/>`

Note: By default, the Worklight JavaScript library auto-hides the splash screen when the application is launched. To have a smooth transition from the splash screen to the web view, set the option **autoHideSplash** to `false` in the `initOptions.js` file and use the WL.App.hideSplashScreen() method to hide the splash screen after all of the page initialization tasks (including loading other JavaScript frameworks) are completed.

Note: Make sure that your application initialization flow does not block the JavaScript call to hide the splash screen. For example, a problem can occur when you set the application to connect to the server on application startup, and you define form-based authentication that waits for the user to enter login credentials. In this case, the application shows a web login form behind the splash screen without a way for the user to interact with it.

Sending actions and data objects between JavaScript code and native code

Send actions and data objects from JavaScript code to native code and from native code to JavaScript code.

The actions are received by action receivers. Actions that cannot be delivered immediately are queued by the Worklight framework and delivered after a suitable action receiver is registered.

Sending actions and data objects from native code to JavaScript code:

IBM Worklight Foundation lets you send actions with optional data objects from iOS or Java code to JavaScript code.

About this task

You might want to send a custom action from native code to JavaScript code (for example, for updating the user interface).

Actions are received by action receivers. Actions that cannot be delivered immediately are queued by the Worklight framework and delivered as soon as a suitable action receiver is registered.

Procedure

1. Add action receiver function in JavaScript code. For example:
   ```
   WL.App.addActionReceiver ("MyActionReceiverId", actionReceiver);
   ```
2. Send action from native code to JavaScript code. For example:
   Android
   ```
   JSONObject data = new JSONObject();
   data.put("someProperty", 12345);
   WL.getInstance().sendActionToJS("doSomething", data);
   ```
   iOS
   ```
   NSMutableDictionary *data = [[NSMutableDictionary alloc] init];
   [data setValue:@"12345" forKey:@"testParam"];
   [[WL sharedInstance] sendActionToJS:@"nativeToJsWithParams" withData:data];
   ```
3. Implement a JavaScript action receiver function to receive and handle incoming actions and data. For example:
   ```
   function actionReceiver(received){
     if (received.action === "doSomething" && received.data.someProperty === "12345"){
       //perform required actions, e.g., update web user interface
     }
   }
   ```

Sending actions and data objects from JavaScript code to native code:

IBM Worklight Foundation lets you send actions with optional data objects from JavaScript to iOS or Java code.

About this task

You might want to send a custom action from JavaScript code to native code, for example, for updating the user interface.

Any object can receive actions. To do so, it must implement the WLActionReceiver interface (for Android) or protocol (for iOS).

Android

```
void onActionReceived (String action, JSONObject data);
```

Example

```
public class MyReceiver implements WLActionReceiver{
  void onActionReceived(String action, JSONObject data){
  //process received action
  }
}
```

Note: Actions are always delivered on a background thread. If you want to update the application user interface from the received action, do so on a main user interface thread, for example by using the Context.runOnUIThread method.

iOS

```
-(void)actionReceived:(NSString*)action withData:(NSDictionary*)data;
```

Example

```
// MyReceiver.h file
#import "WLActionReceiver.h";
@interface MyReceiver: NSObject <WLActionReceiver>{}
@end

// MyReceiver.m file
@implementation MyReceiver
-(void)onActionReceived:(NSString *)action withData:(NSDictionary *)data{
    // process received action
}
@end
```

Note: Actions are always delivered on a background thread. If you want to update the application user interface from the received action, do so on a main user interface thread, for example by using the performSelectorOnMainThread method.

Actions are received by action receivers. Actions that cannot be delivered immediately are queued by the Worklight framework and delivered as soon as a suitable action receiver is registered.

Procedure

1. Add an action receiver in native code. For example:

   Android

   ```
   WL.getInstance().addActionReceiver(myReceiver);
   ```

   iOS

   ```
   [[WL sharedInstance] addActionReceiver:myReceiver];
   ```

2. Send action from JavaScript code to native code. For example:

   ```
   WL.App.sendActionToNative("doSomething", { customData: 12345} );
   ```

3. Implement the native onActionReceived. For example:

   Android

   ```
   void onActionReceived(String action, JSONObject data){
    if (action.equals("doSomething")){
    //perform required actions, e.g., update native user interface
    }
   }
   ```

   iOS

```
-(void) onActionReceived:(NSString *)action withData:(NSDictionary *) data {
    if ([action isEqualToString:@"doSomething"]){
        // perform required actions, e.g., update native user interface
    }
}
```

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 Foundation, 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 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" on page 480 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 4, "Tutorials and samples," on page 35.

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, minimum-
```

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 Foundation 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.

IBM Worklight Foundation supports the Dojo 1.9.3 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 482.
- 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 508.

- Use predefined application templates to speed up the development of your application. For more information, see "Mobile patterns" on page 519.
- 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 483.
- For information about how to use Dojo to create a globalized Worklight application, and how to achieve this process by using Dojo Mobile, see "Developing globalized hybrid applications" on page 932.
- 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 496.

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-1.4.2.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 508.

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:

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 Worklight application.

A 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:

Special view displays Dojo modules actually requested by the application
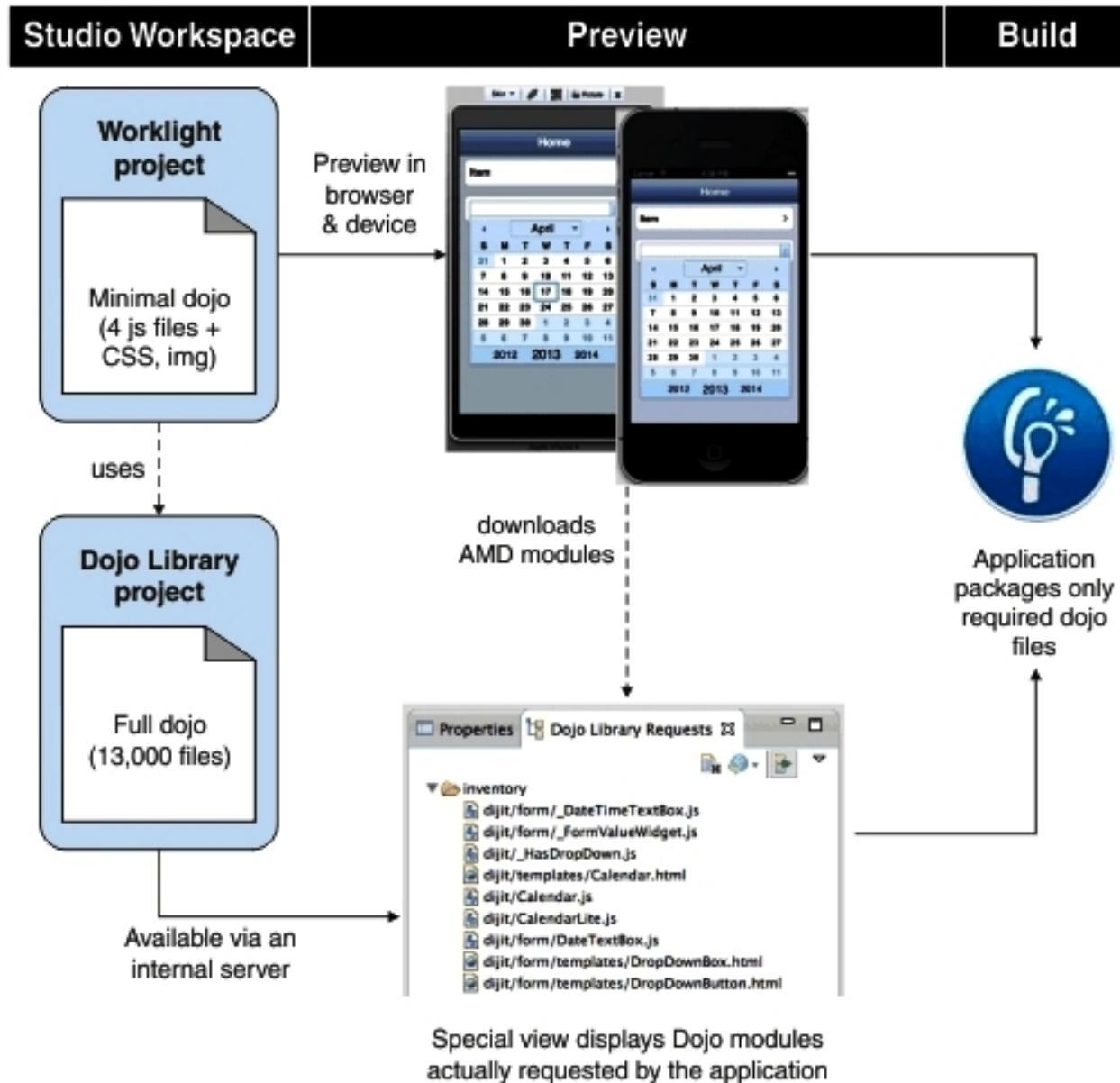
To demonstrate, here are illustrations that show the start of a new Worklight project. After you click **Configure JavaScript Libraries**, a wizard opens where you must click **Add Dojo Toolkit**. An extra field in the template is displayed called **New Dojo Library**.
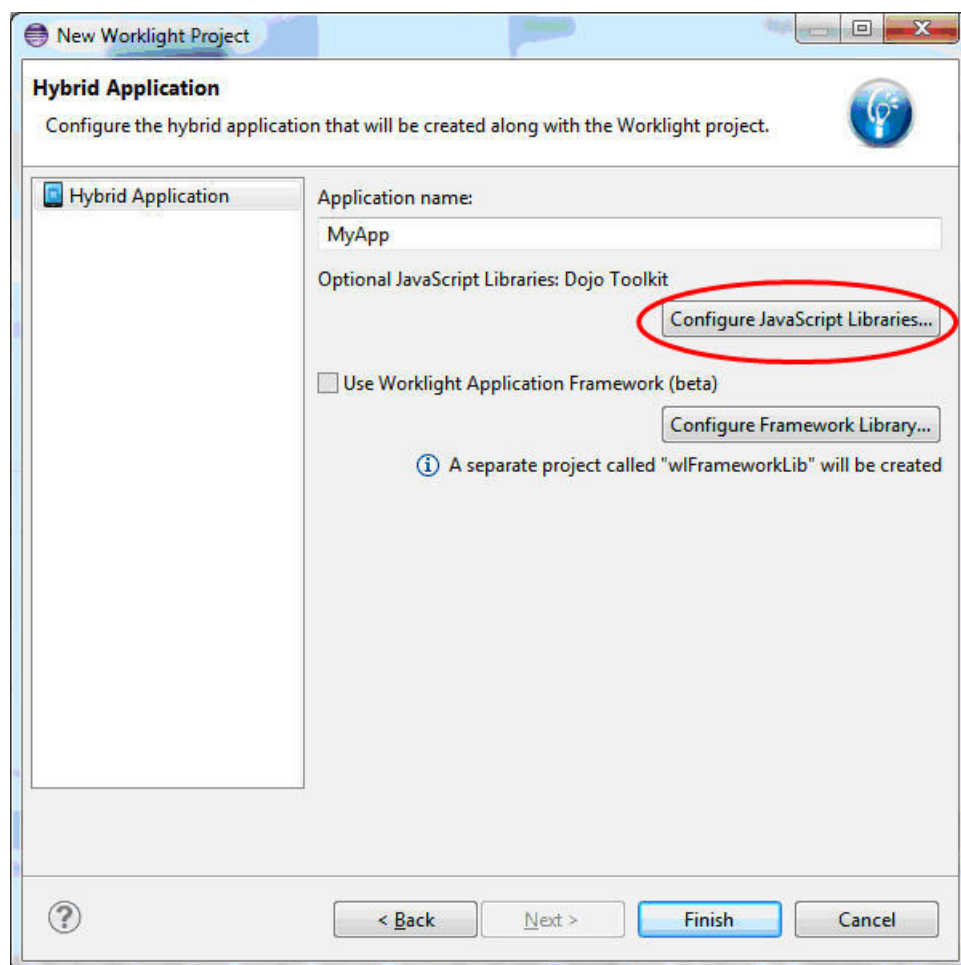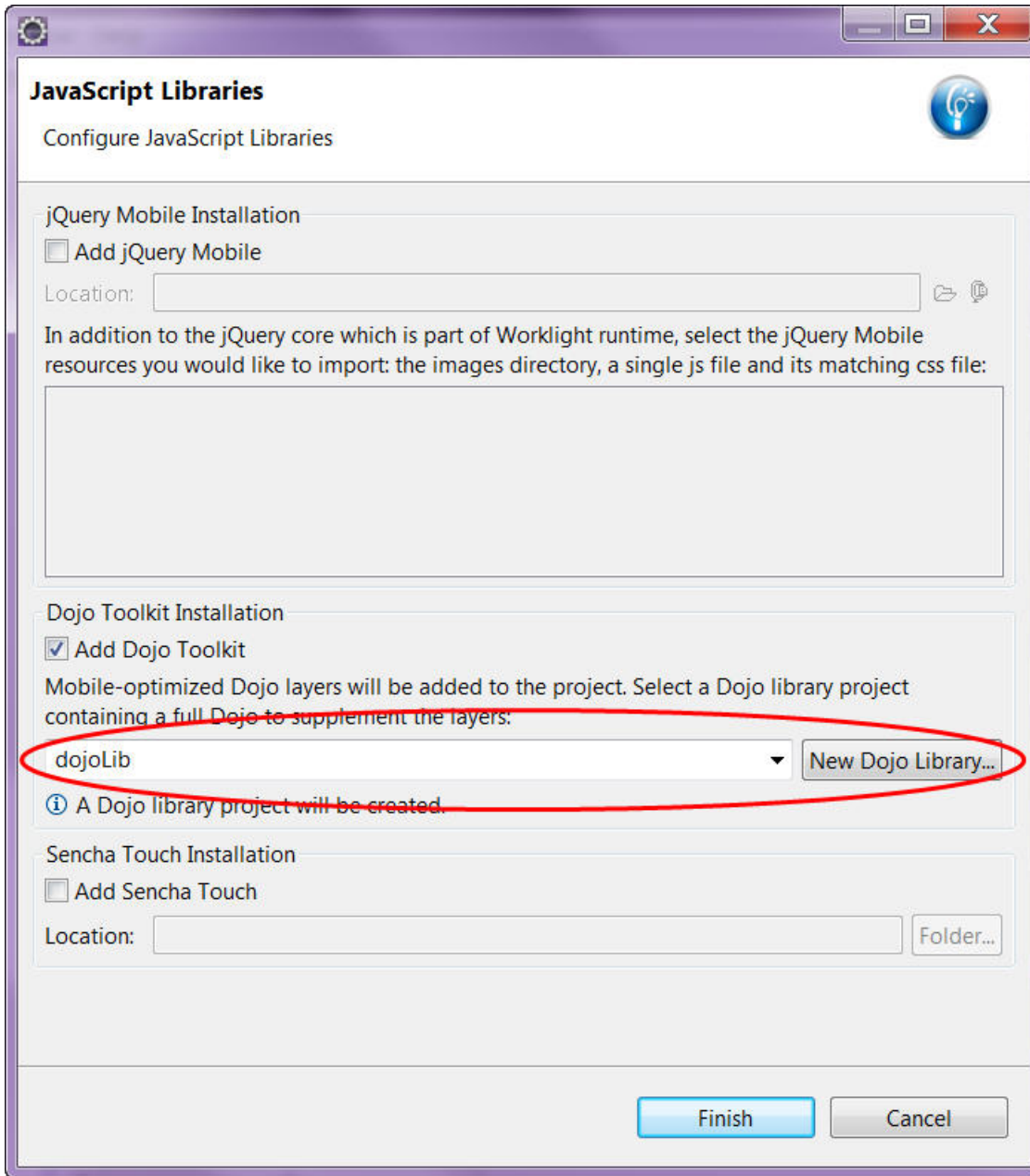
Note: You can create multiple libraries each for a different version of Dojo. A Worklight project is linked to one library.

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.
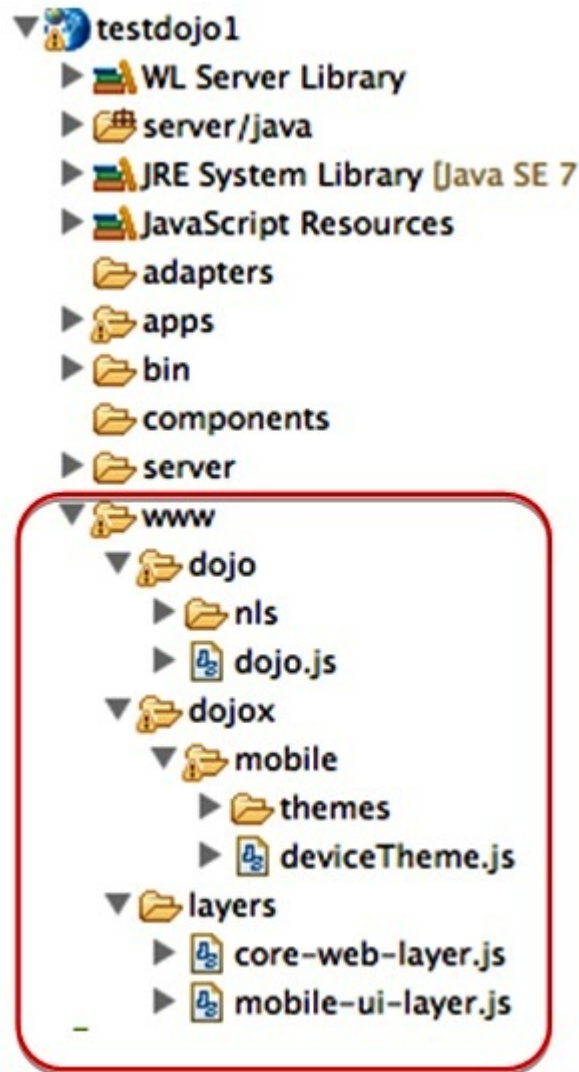
You can manually set up more themes. First copy a theme into the www folder, and then set up the project css settings. Dijit widgets require a theme, Dojox widgets each bundle their own theme css.
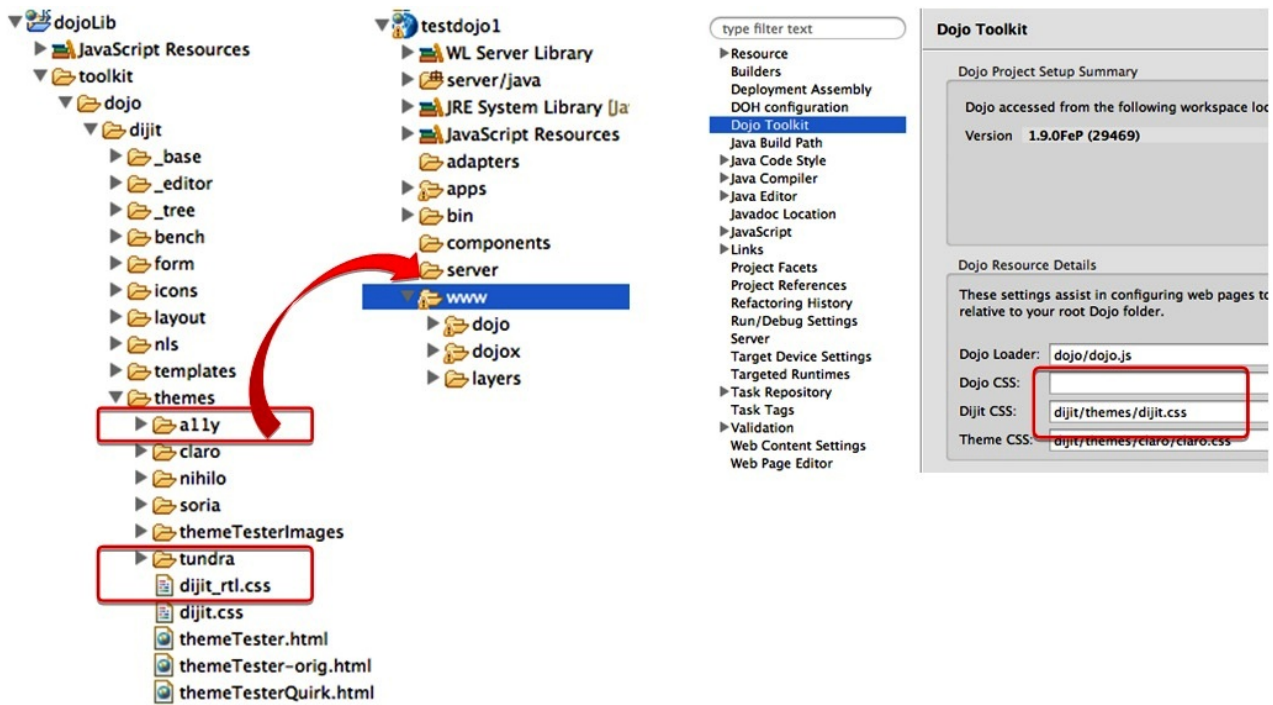
You can start coding your application. Use any Dojo modules (you no longer need to consider what files to import into the project). In this example, the "DateTextBox" comes from `dijit/form` but this module and its dependencies are

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"></sc
<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/
```

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.
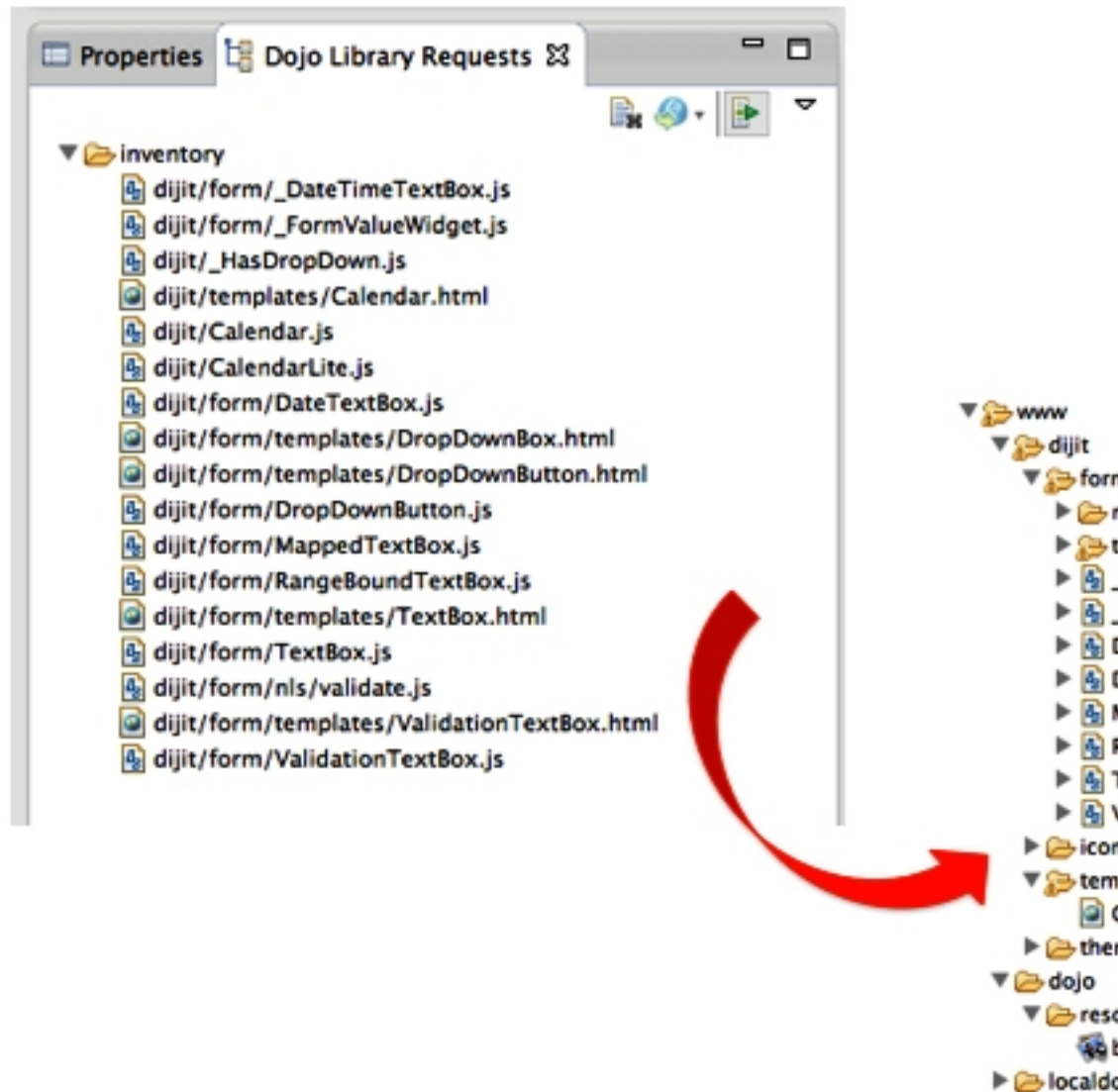
You must turn off the library and verify all the Dojo files are present in the application. Turn off the server

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);
```
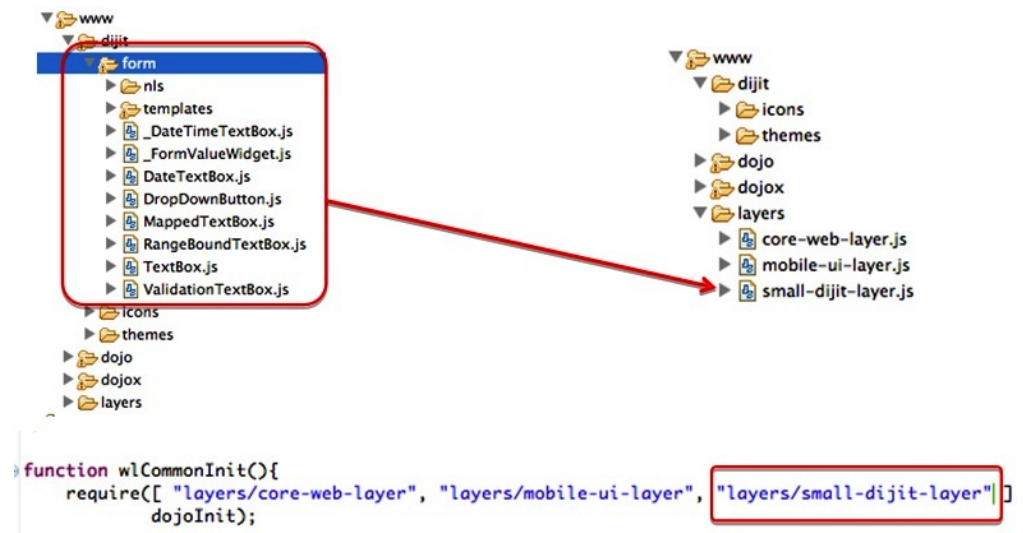
The IBM Worklight Studio tools use the Dojo that is contained in the Worklight project and the associated Dojo Library project. The following Worklight Studio tools use the Dojo library content:

Rich Page Editor
> The Rich Page Editor displays all of the widgets that are available in the Dojo library.
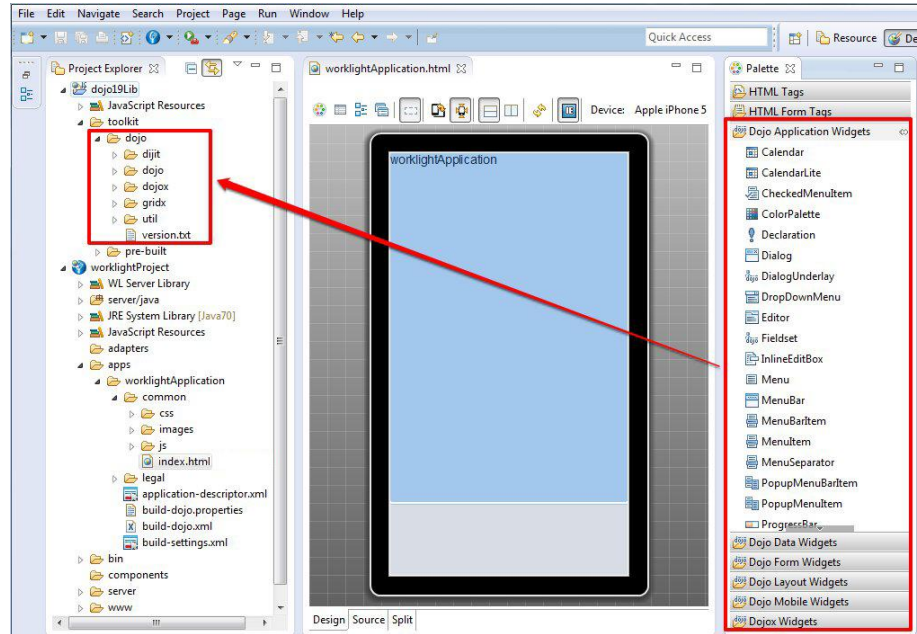
You can explore and test various Dojo artifacts. You could run and test your application outside of the Dojo library project. If you test on an external device or emulator, IDE must be running and must have Internet connectivity.

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 496.
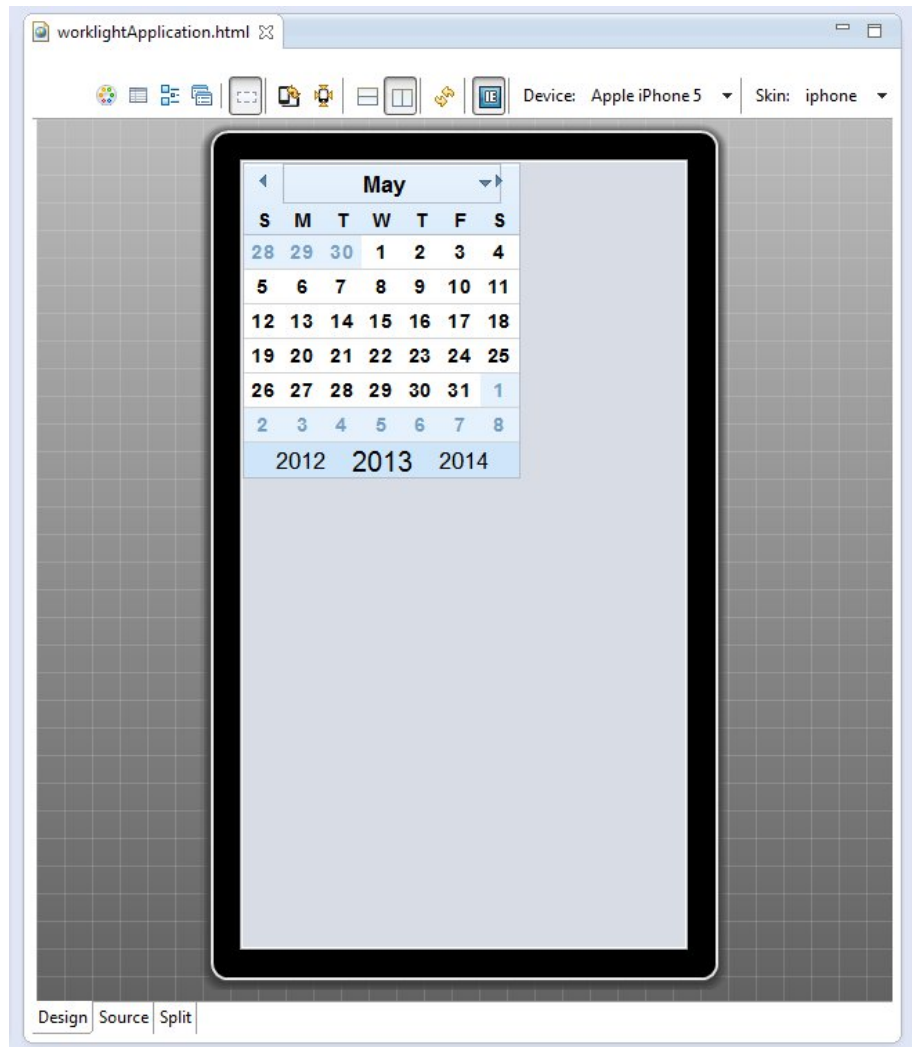
JavaScript source validation and content assist

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.
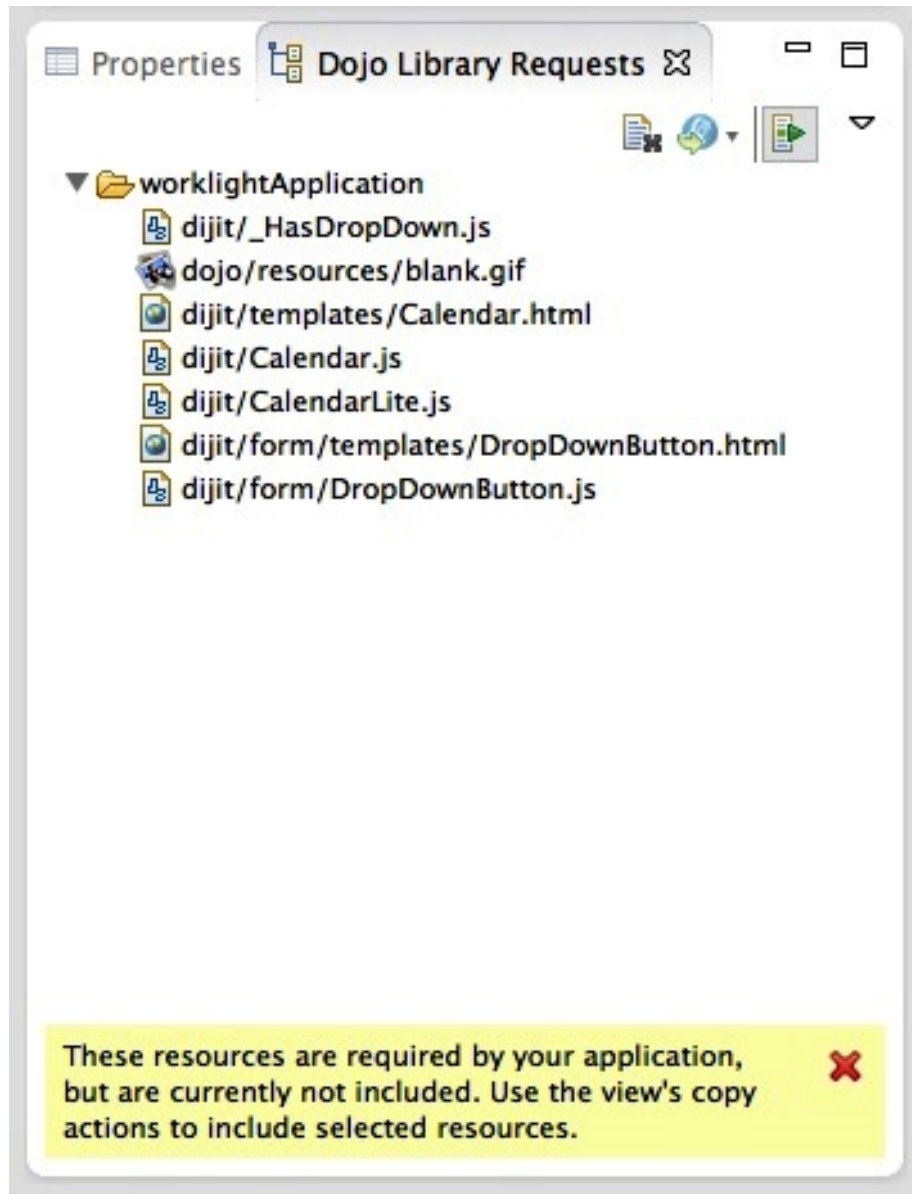
If the missing resources are required by the final Worklight application, you must add all of the missing resources to the Worklight project. The resources that are logged in the Dojo Library Requests are not available outside of the Worklight Studio development environment.

To add the missing resources to your application, the view provides two copy actions.

The **Copy to Project** action ![icon] 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 ![icon] 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 ![icon] , 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 Worklight project. The Dojo Console view helps you determine which resources were provided by the Dojo library project. Only the resources in thewww 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 483 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 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 Foundation to a new version of Dojo that is included with IBM Worklight Foundation. If you want to take advantage of a version of Dojo in open source that is not yet in IBM Worklight Foundation, 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.:
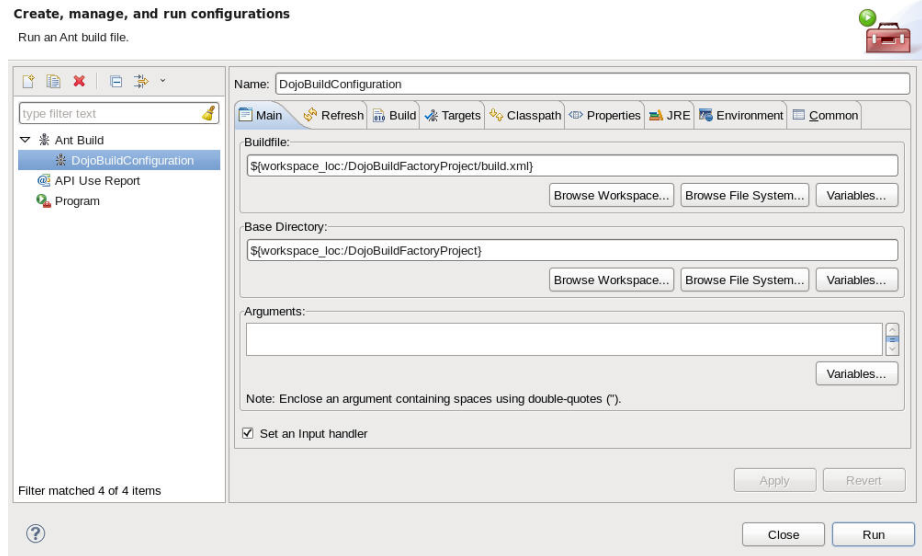
e. Go to the Properties tab and confirm that **Use global properties as specified in the Ant runtime preferences** is cleared.

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`
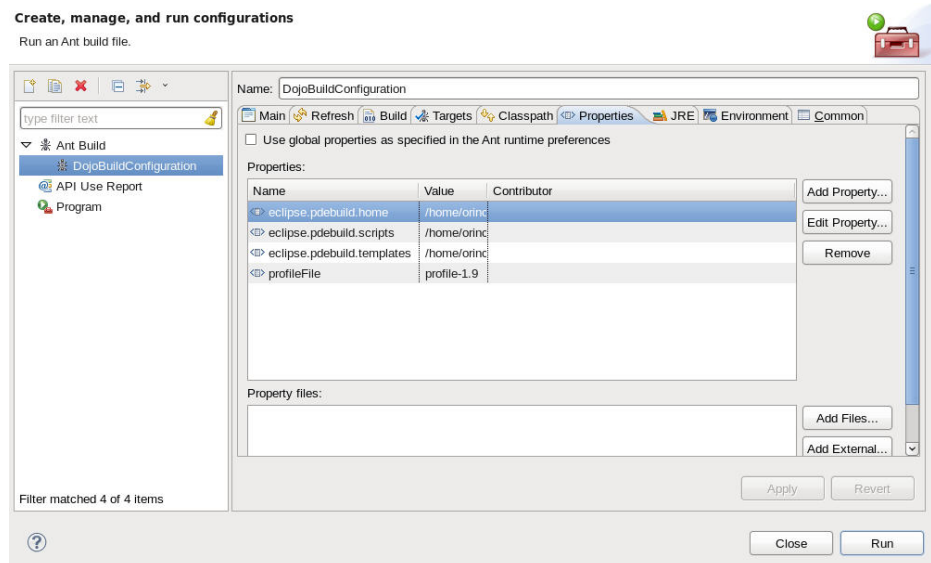
g. After you set up the configuration, click **Run**. The build takes about 6-15 minutes and can be monitored in the console view.

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 theprofile-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:

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.

f. Click **Finish** to close this dialog. Confirm that the properties page indicates it uses the new Dojo Library, then click **OK**.
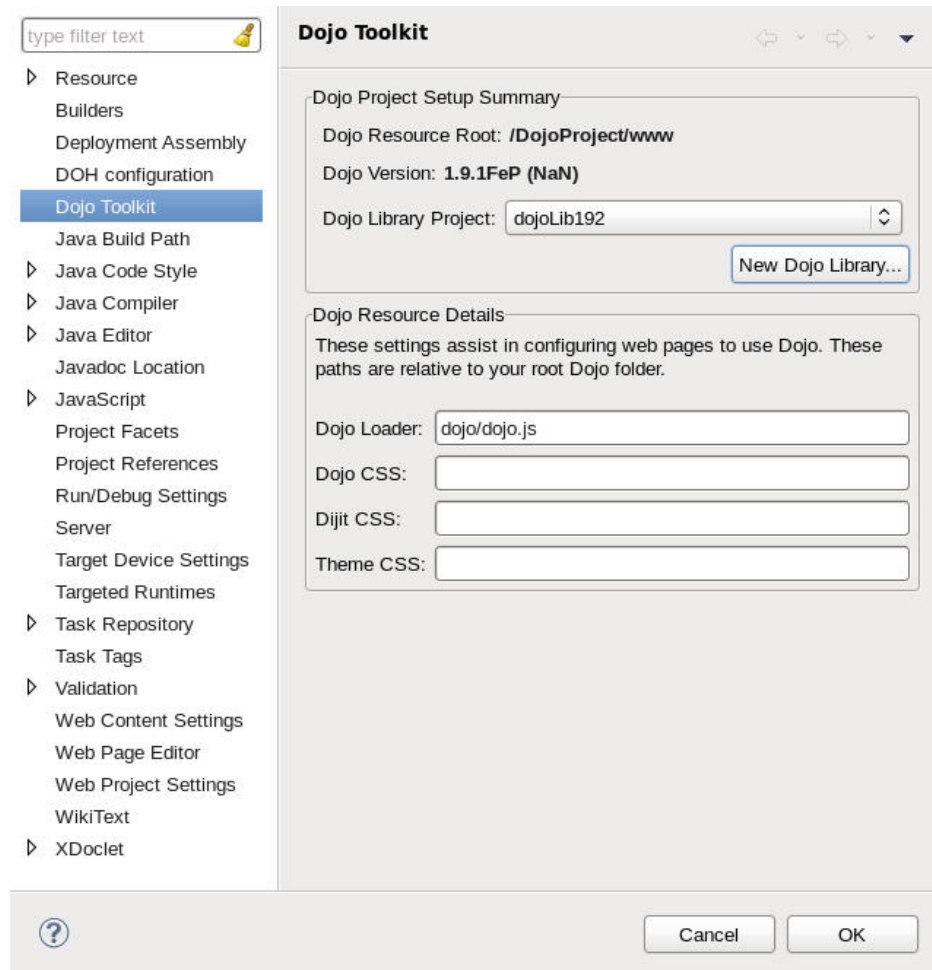
A new Dojo Library project must be created, but it is still incomplete because it lacks the optimized layers you created previously, so there are still few steps to complete.

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.
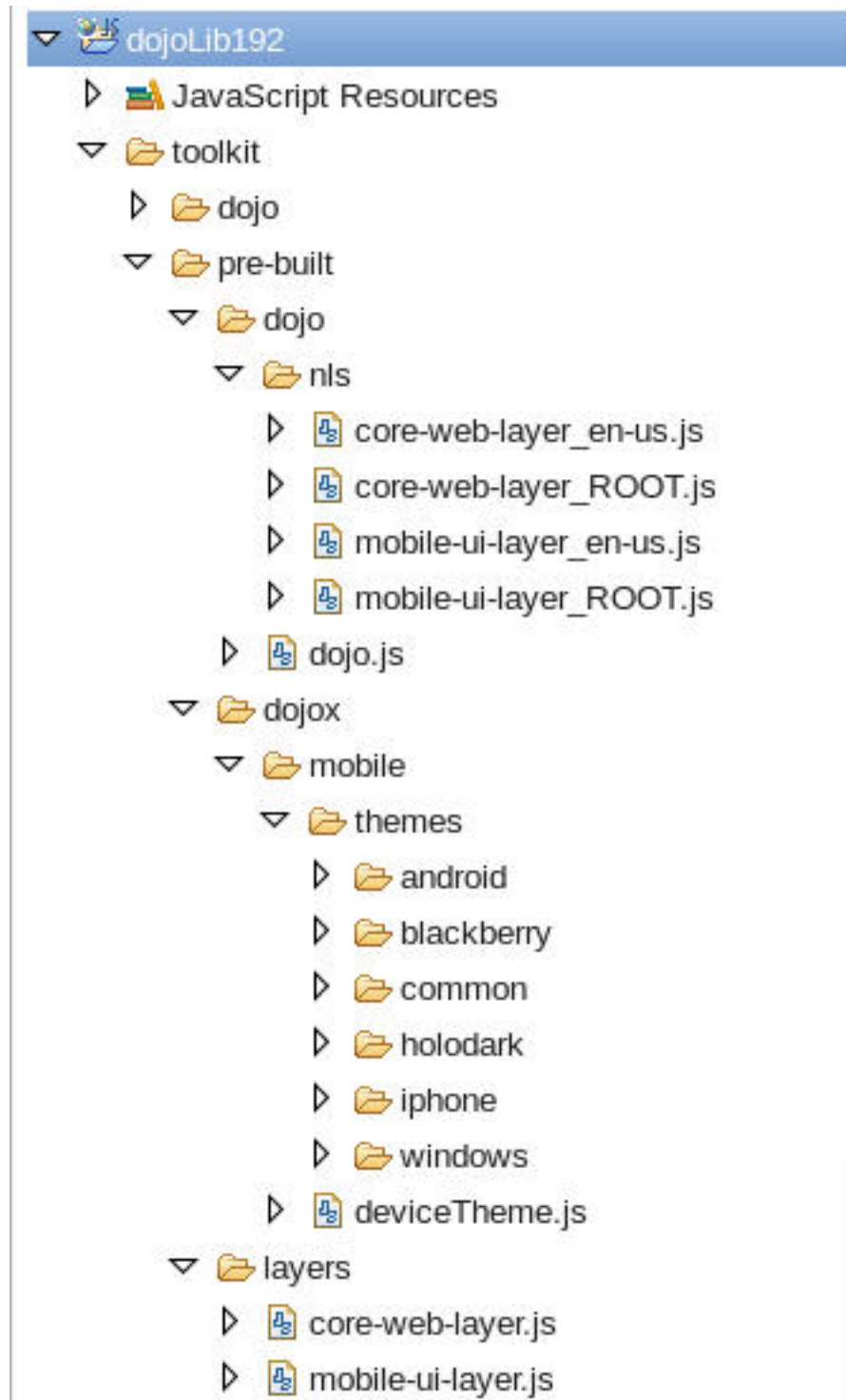
Copy the files from the `result/compressed` folder in the Dojo Build Factory project.

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 483.
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 Foundation. 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.
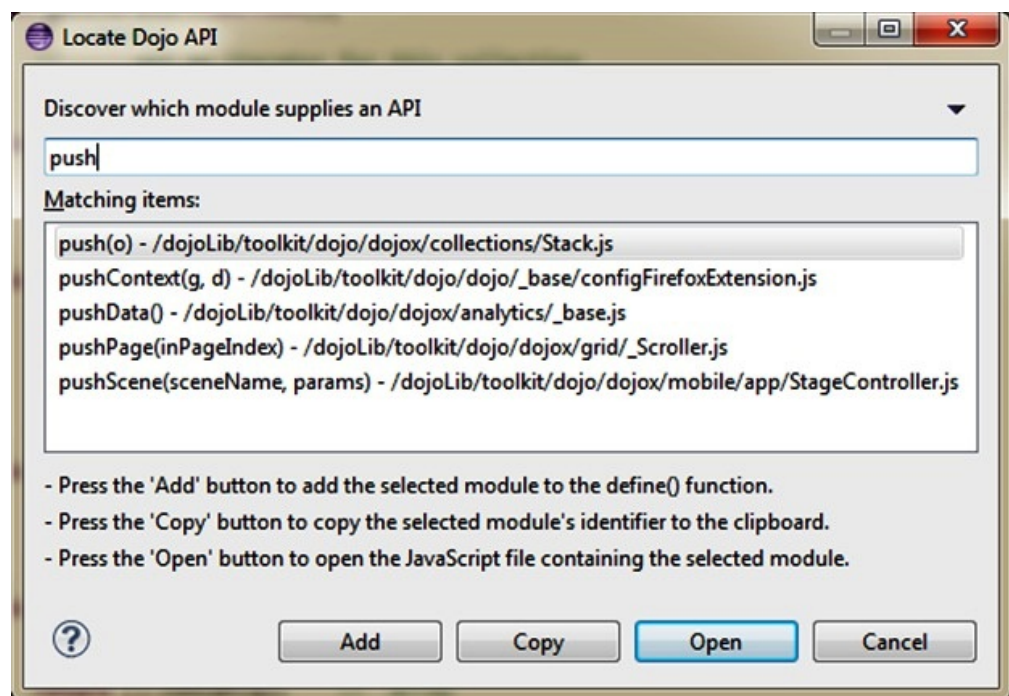
Application skins

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 use Worklight Studio to define a skin, Worklight Studio generates a folder for the skin resources and adds a `<skin>` element in the application descriptor file. The `<skin>` element includes the name of the skin and a list of resource folders. When Worklight Studio builds the application, it applies the optimization rules on the resource folders in the order in which 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.
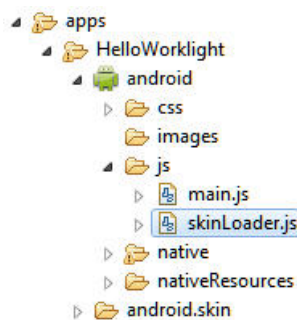
Figure 37. The `skinLoader.js` file

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 Foundation, you can create a settings page to change the URL of the Worklight Server.

With IBM Worklight Foundation, 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 that belong 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.

By default, the settings page is disabled with the `include` attribute of `<worklightSettings>` set as `false` in the relevant environment element of the `application-descriptor.xml` file.

To activate the settings page for the supported environments, change the `include` attribute of `<worklightSettings>` to `true`. 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 93. Rich Page Editor views

Editor view	Description
Source	The Source view helps you to view and work directly with the source code of a file. The Mobile Navigation, Palette, Outline, Page Data, and Properties views have features that supplement the Source view.
Split	The Split view combines the Source and Design views in a split screen view. Changes that you make in one part of the split screen are automatically updated in the other part. You can split the view horizontally or vertically.
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.
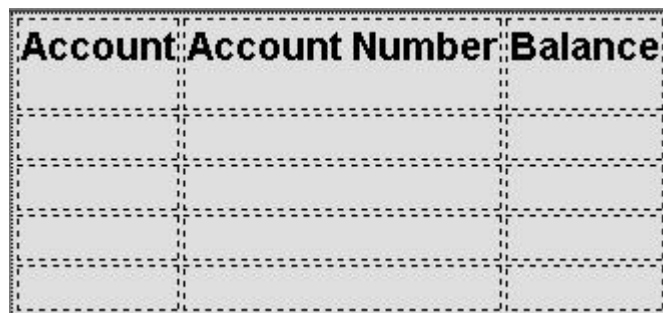
The following screen capture shows what the same table looks like in the Design view of Rich Page Editor when Design Mode is not enabled.

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:
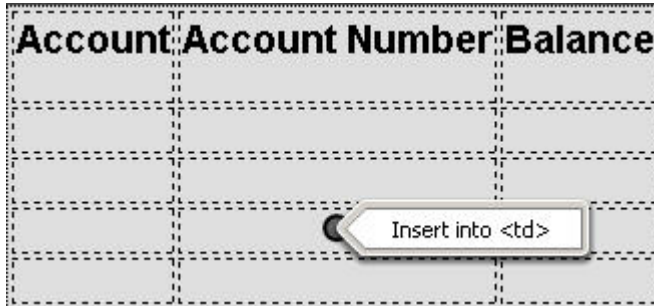
Selecting a cell in a table opens a pop-up cue that you can use to add a column or row:
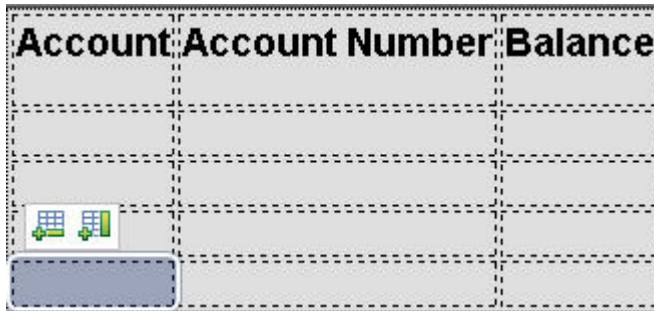
Browser requirements for Rich Page Editor:

Rich Page Editor uses embedded browsers to produce a visual representation of a web page in the Design view. The browsers that are available in Rich Page Editor and their installation requirements vary according to the platform.

Procedure

The following table lists and describes the supported browsers in Rich Page Editor, by platform:

Platform	Supported browsers
Windows	**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 Foundation 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 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 Foundation 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 94. 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 94. 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 Creating Mobile Pattern Projects.

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 519
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.

Creating mobile pattern projects:

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 wizard to create your own pattern project.

    a. Click **File** > **New** > **Project**.

    b. Expand the **Web** folder and select **UI Pattern Project**.

    c. Click **Next**.

    d. Give your UI Pattern Project a name.

    e. Optional: You can click **include jQuery** and **add jQuery Resources** to the project.

      Note: Files are **jQuery Mobile** (JS and CSS files) and **jQuery JS Core** so you can properly preview the app by using the Rich Page Editor.

    f. Click **Finish**. A UI Pattern Project is created.

2. Right-click your UI Pattern Project and select **New** > **UI Pattern**.

    Note: Your project must contain either Dojo framework or jQuery framework, or both, for you to be able to continue in the **UI Pattern** wizard.

3. Define the name of your pattern and click **Finish**. A folder with the name of your new pattern is added to the `WebContent` folder. This folder contains the pattern's resources.

4. Open the `pattern.html` file that is found in one of the following locations: `WebContent/`*`pattern_name`*`/Dojo` or `WebContent/`*`pattern_name`*`/jQuery`.

You can create your pattern by editing this file in Rich Page Editor in either of the following ways:

- For Dojo:

  Add the following two script tags to the `pattern.html` file.

  ```
  <script type="text/javascript" pattern.discardNode="true">
    require([ "dojox/mobile/parser", "dojox/mobile/compat" ]);
  </script>
  ```

  This script tag is required to preview the pattern in the Mobile Pattern Browser. The `pattern.discardNode` attribute is used by patterns to identify when an element is discarded from pattern insertion.

  ```
  <script type="text/javascript">
    require([ /*START_DEPENDENCIES*/ "dojox/mobile/ScrollableView" /*END_DEPENDENCIES*/ ]);
  </script>
  ```

  In the second script tag, when a pattern is added to a mobile page, the `require` elements between the `START_DEPENDENCIES` comment and the `END_DEPENDENCIES` comment are translated to Dojo module requests. Then, they are inserted into the Dojo `require` section in the final application. The `require` elements outside the `DEPENDENCIES` comments are not added to the final application. When you add a Dojo widget to your mobile pattern, add the necessary modules for that widget between the `DEPENDENCIES` comments.

  Add a scrollable view widget to the pattern. Inside of your new view widget is where you can add the rest of your pattern widgets.

- For jQuery:

  Add a page widget to the pattern. Inside of your new page widget is where you can add the rest of your pattern widgets.

5. Save your UI Pattern project. You can add the mobile pattern to a Dojo or jQuery project by creating or opening an IBM Worklight hybrid project with Dojo or jQuery support. Open the `index.html` file that is found under `apps/`*`app_name`*`/common` in Rich Page Editor.

6. Click + from the Mobile Navigation view. Use the following figure for guidance:

Figure 38. Mobile Navigation view

If the Mobile Navigation view is not visible, click **Window** > **Show view** > **Other**, expand the **Web** folder, select **Mobile Navigation**, and click **OK**.

7. Complete the Add jQuery Mobile Page or Add Dojo Mobile Page wizard.

   a. Type the name of your UI Pattern in the **Id** field.

   b. Select the **Create from UI pattern** option.

   c. From the **Pattern Set** menu, select *UI_Pattern_project_name*. Your mobile pattern is displayed in the Mobile Pattern Browser.

   d. Select your mobile pattern and click **Finish**.

   Your new UI Pattern is displayed in the preview.

8. Save your Dojo or jQuery project where the pattern was added.

Results

The new pattern is available for you to use with other Dojo and jQuery projects in the workspace.

Related tasks:

"Adding a mobile pattern to an application" on page 520
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 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:

html ▶ body ▶ div ▶ div ▶ ul ▶ li ▶ a
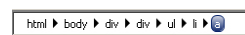
You can scroll through the breadcrumb trail without losing the position of your cursor in Rich Page Editor. Using this feature, you can quickly update the properties of ancestor elements.

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.
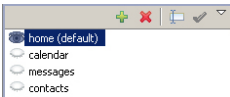
External mobile view or page
> A mobile view or page that is written in a separate file or fragment. Creating mobile views or pages in separate files or fragments makes source code shorter and easier to manage.

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.
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**.

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, BlackBerry 10, Windows Phone 8, and mobile web application environments.

Tip: When you preview a Worklight application on an Android, iPhone, iPad, BlackBerry 6 and 7, BlackBerry 10, 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, BlackBerry 10, 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.
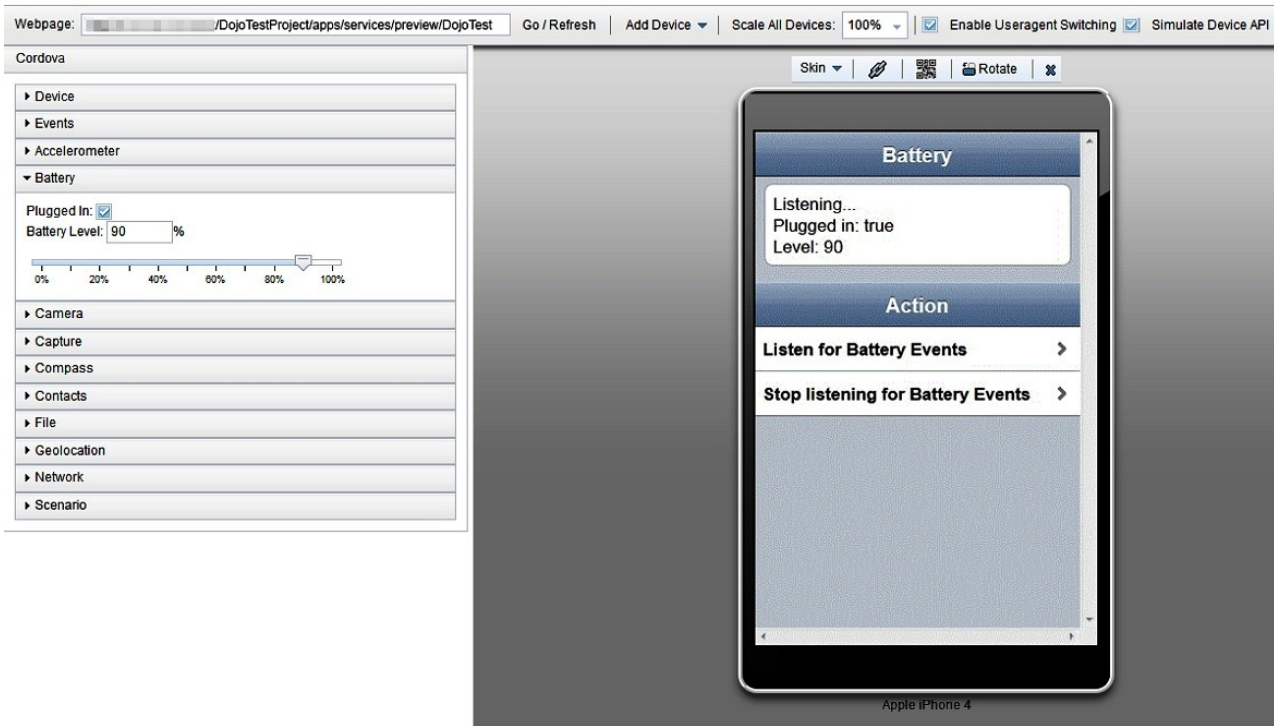
Testing mobile applications with the Mobile Browser Simulator:

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 517

2. Enable user agent switching.

Procedure

In the simulator, click the device list and then select the device that you want to simulate.
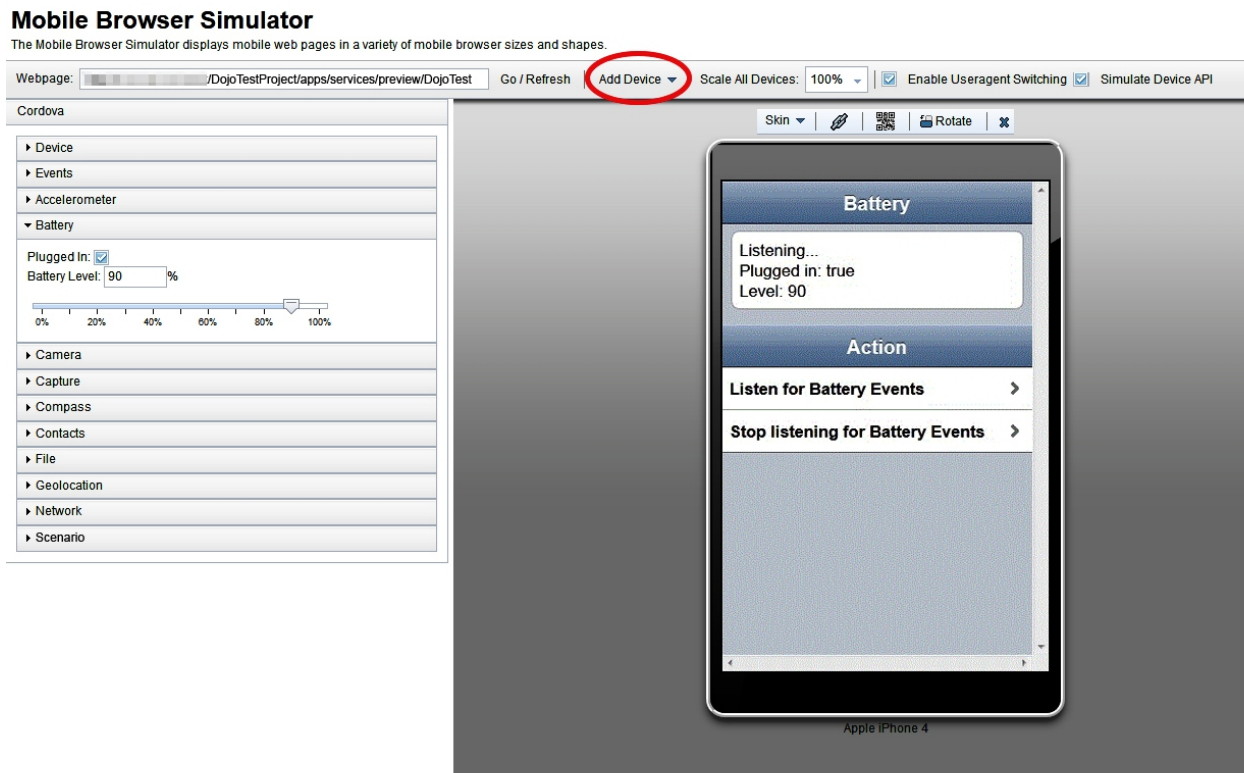
Adding devices:

Before you begin

To view your web application in the simulated devices by using the appropriate style sheets, ensure that these tasks are completed:

1. "Creating web pages for mobile devices" on page 517

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.

Calibrating the Mobile Browser Simulator:

Since browsers cannot accurately paint physical dimensions, you must calibrate the Mobile Browser Simulator.

Before you begin

Test your application using the Mobile Browser Simulator.

Procedure
1. From the **Scale All Devices** list, select **Physical device size**.
2. Click **Calibrate Physical Size** to open the Physical Size Calibration dialog.
3. Follow the instructions in the dialog to calibrate your Mobile Browser Simulator. After you complete all of the steps in the dialog, close the dialog.

Enabling user agent switching:

You can use the Mobile Browser Simulator to render your web applications on different mobile devices. To render your web applications with the appropriate style sheets and theme, you must enable user agent switching.

Before you begin
- Enable the detect device option when you create your web page.
- Test your application by using the Mobile Browser Simulator.

About this task

The Useragent Switcher Extension is a browser extension that provides the user agent switching feature. The Mobile Browser Simulator supports implementations of this browser extension for the following web browsers:
- Mozilla Firefox.
- Chrome 17 and later, with limitations.

Useragent Switcher Extension for Chrome

> The Useragent Switcher Extension emulates requests from different device-specific agents. When a web application checks the user agent on the server to create content, it is correctly simulated.

> The Useragent Switcher Extension includes support for Dojo Mobile 1.7 and later. If you enabled the detect device option when you created your Dojo Mobile page, the Useragent Switcher Extension uses the automatic device detection and theme loading for Dojo Mobile to select the appropriate theme.

Procedure
1. Click **Enable Useragent Switching**.
2. If the latest version of the Useragent Switcher Extension is not installed, the Install Useragent Switcher Extension dialog opens. Click **Install Browser**

Extension.

Mobile Browser Simulator

The Mobile Browser Simulator displays mobile web pages in a variety of mobile browser sizes and shapes.
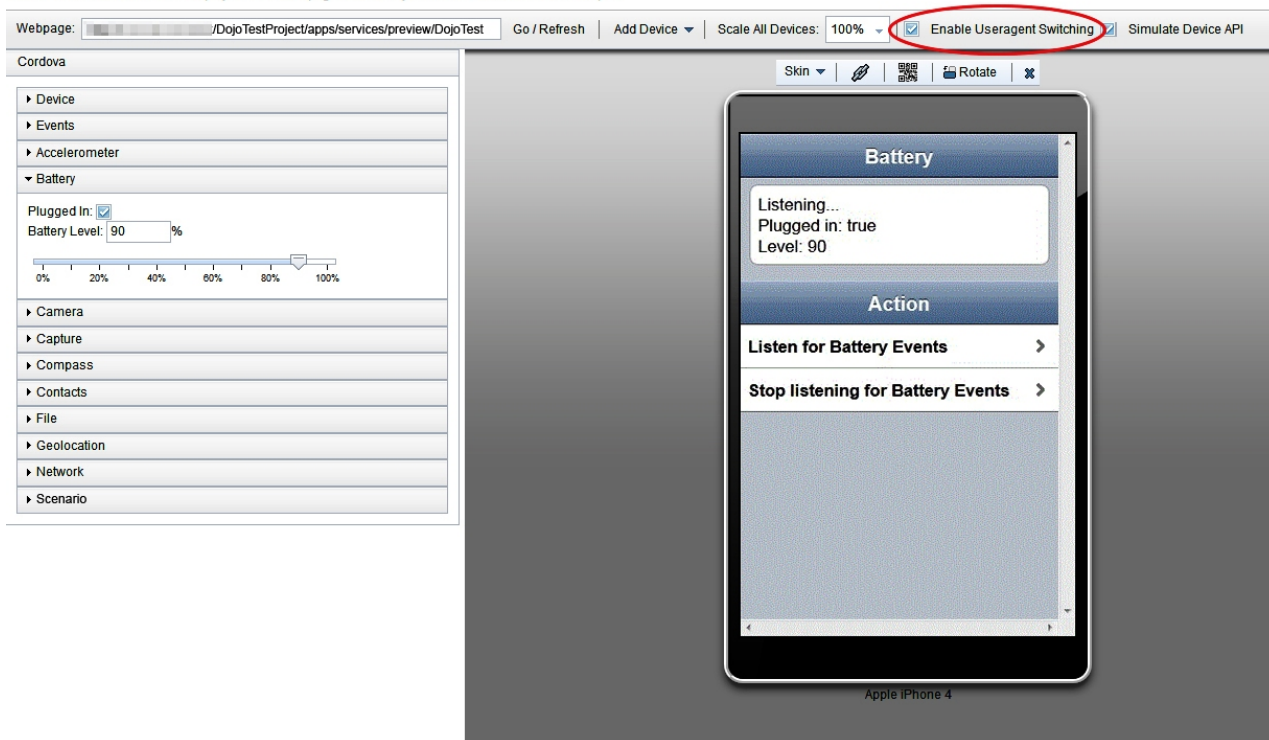
Results

You can now view your web application with the appropriate style sheets and theme in the simulated mobile devices.

Previewing your Worklight applications

You can use the Mobile Browser Simulator to preview Worklight applications on iPhone, iPad, Android phones and tablets, BlackBerry 6 and 7, BlackBerry 10, Windows Phone 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 environments

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**.

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:
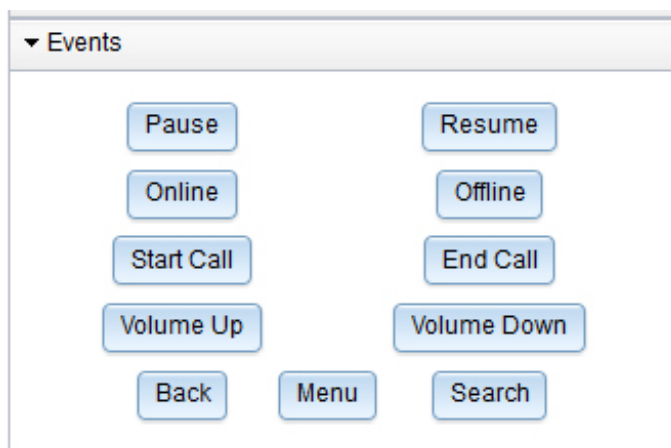
Accelerometer

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**.

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**.
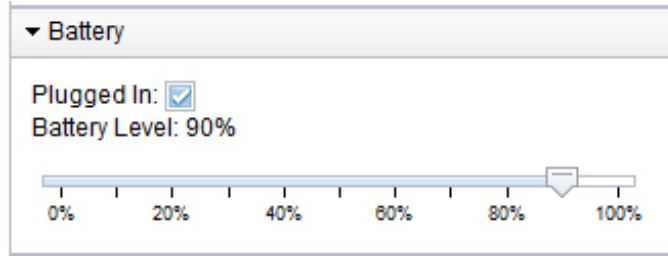
Camera

Specifies which image to use for the camera and for the album:

- Simulate a photo taken with the camera (Camera.sourceType == Camera.PictureSourceType.CAMERA)
- Photo from the device photo album or library (Camera.sourceType == Camera.PictureSourceType.PHOTOLIBRARY or Camera.sourceType == Camera.PictureSourceType.SAVEDPHOTOALBUM)

To change the size of the selected photos, click **XS**, **S**, **M**, **L**, or **XL**.
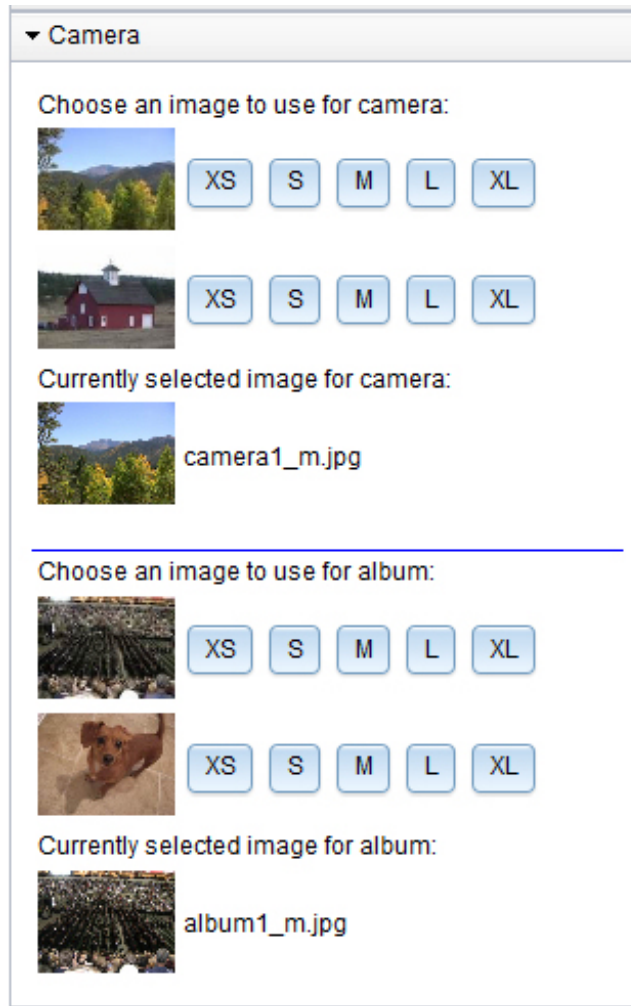
Capture

Simulates the Cordova capture API by using the following methods:

- capture.captureAudio
- capture.captureVideo

You can select the audio and video recordings that you want to use, and play these recordings by using the HTML5 players.
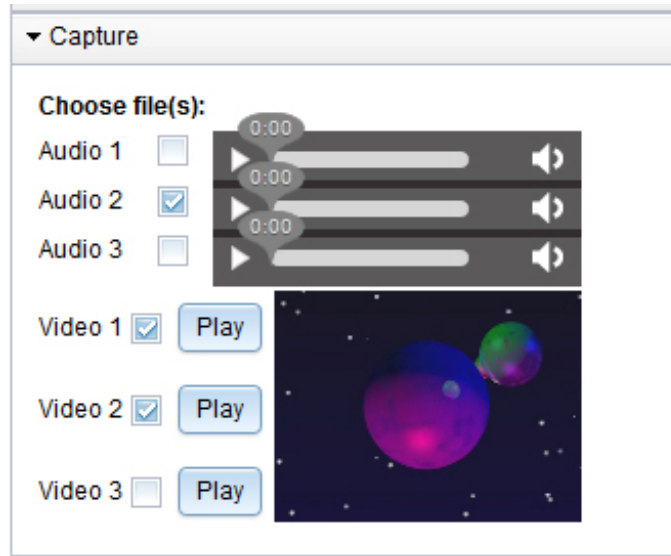
Note: The Capture section is available on both Mozilla Firefox and Google Chrome. For improved support of the HTML 5 players, upgrade these browsers to the latest version.

Compass

Defines the values returned by the Cordova API when querying Compass data. To generate a new set of values, click **Next**. To generate the values periodically, click **Start**. You can also set the compass values by directly interacting with the compass widget.

Contacts

Shows the available contacts for the mobile device. You can delete contacts and refresh the list of available contacts.
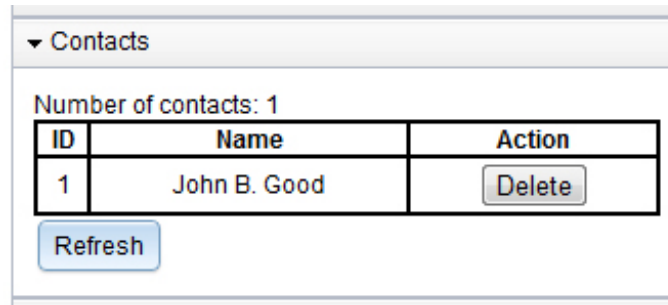
To create new contacts for the mobile device, use the Cordova Contacts API from your simulated mobile web page. The contacts are stored in the Web SQL Database which is supported by default by Google Chrome and Safari. To simulate the Contacts API with Firefox, you must install an Add-on in your browser that adds basic WebSQL support to Firefox.

File   Simulates the Cordova File API by running an applet. To update the display of the file system that you can access through the Cordova API, click **Refresh**. Use the Cordova API to access this file system to read and write.
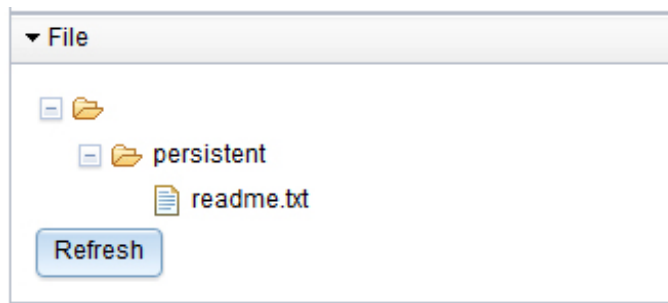
Double click anywhere on the file tree to open a file viewer. The file viewer allows for manual manipulation of your simulated file system.

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**.

Network

Defines the active connection of the device.

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.

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.

Testing hybrid location service applications

You can use the Mobile Browser Simulator 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 Mobile Browser Simulator 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 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.

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 962.

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 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 962.

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.
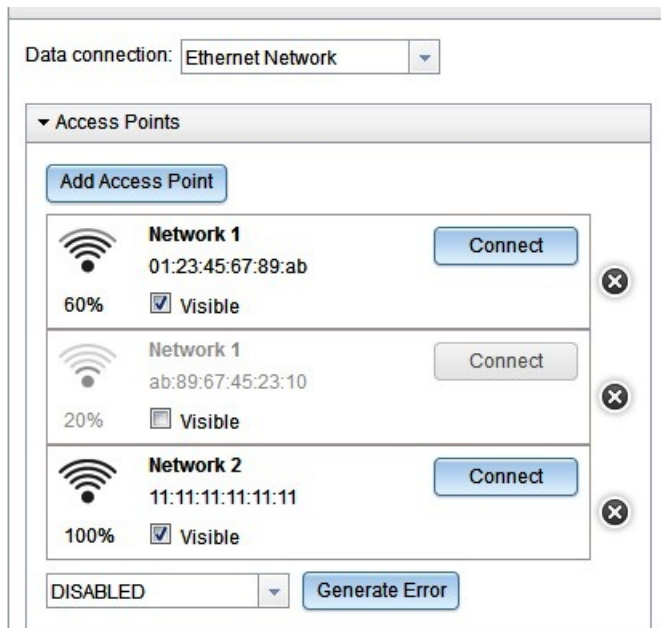
The **Visible** check box indicates whether the access point is visible to the device, and whether it could be returned by a call to `WL.Device.Wifi.acquireVisibleAccessPoints`.

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.
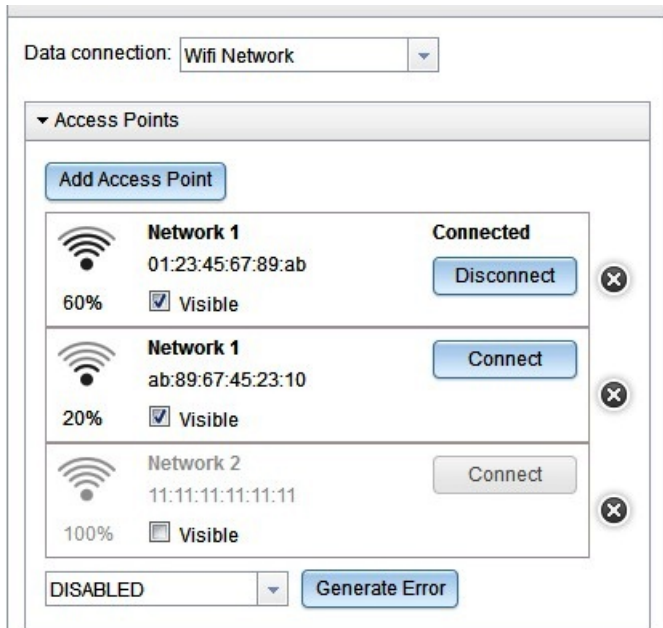
Only one access point can be connected at a time. The access point is connected to the data network, which switches to WiFi. When you switch the network to something that is not WiFi, then the connected access point is disconnected.

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 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 allows developers to quickly test, debug, and experiment with their hybrid applications from the convenience of the web browser. Although Mobile Browser Simulator 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 Mobile Browser Simulator. 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 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 39.

- 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.
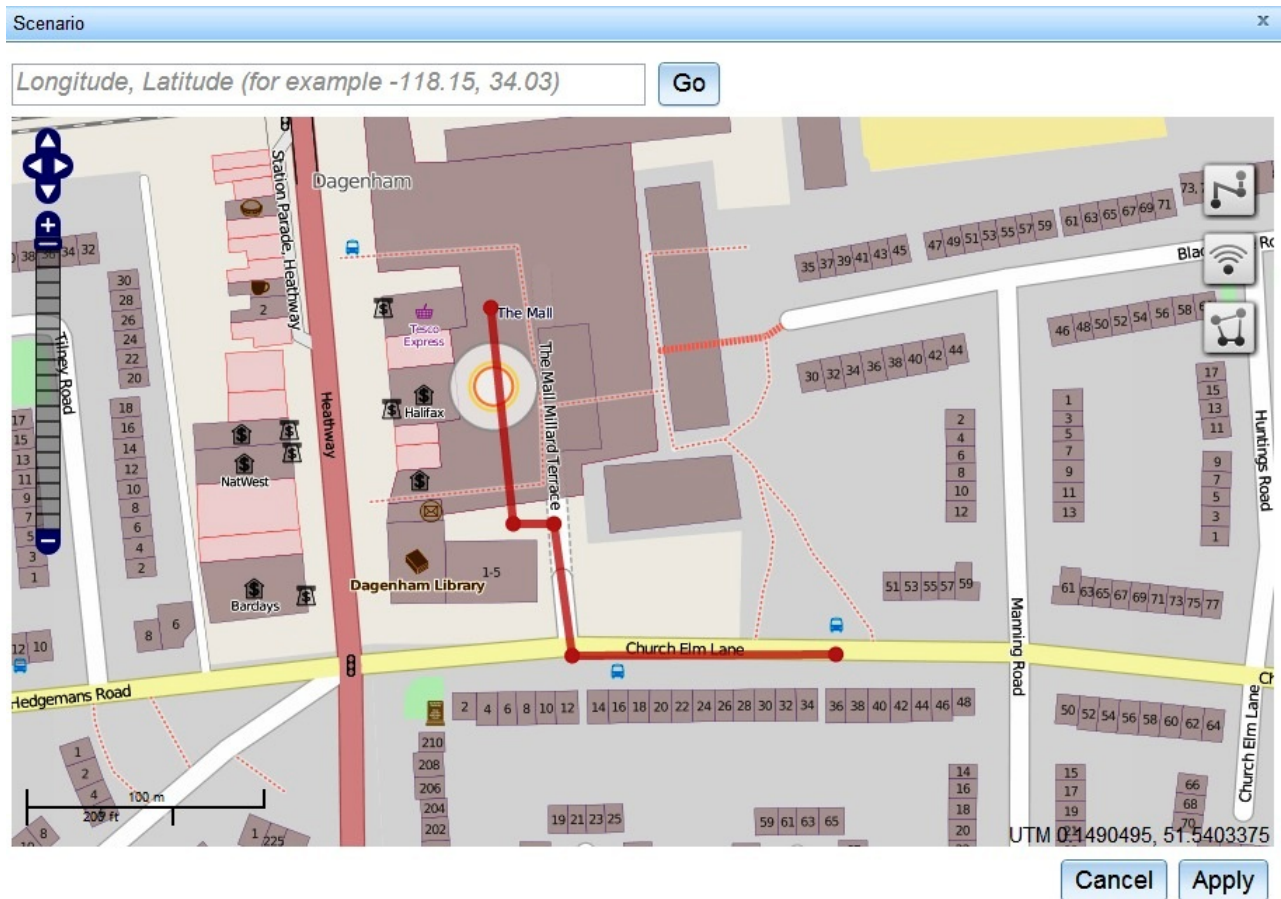
Figure 40.

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.
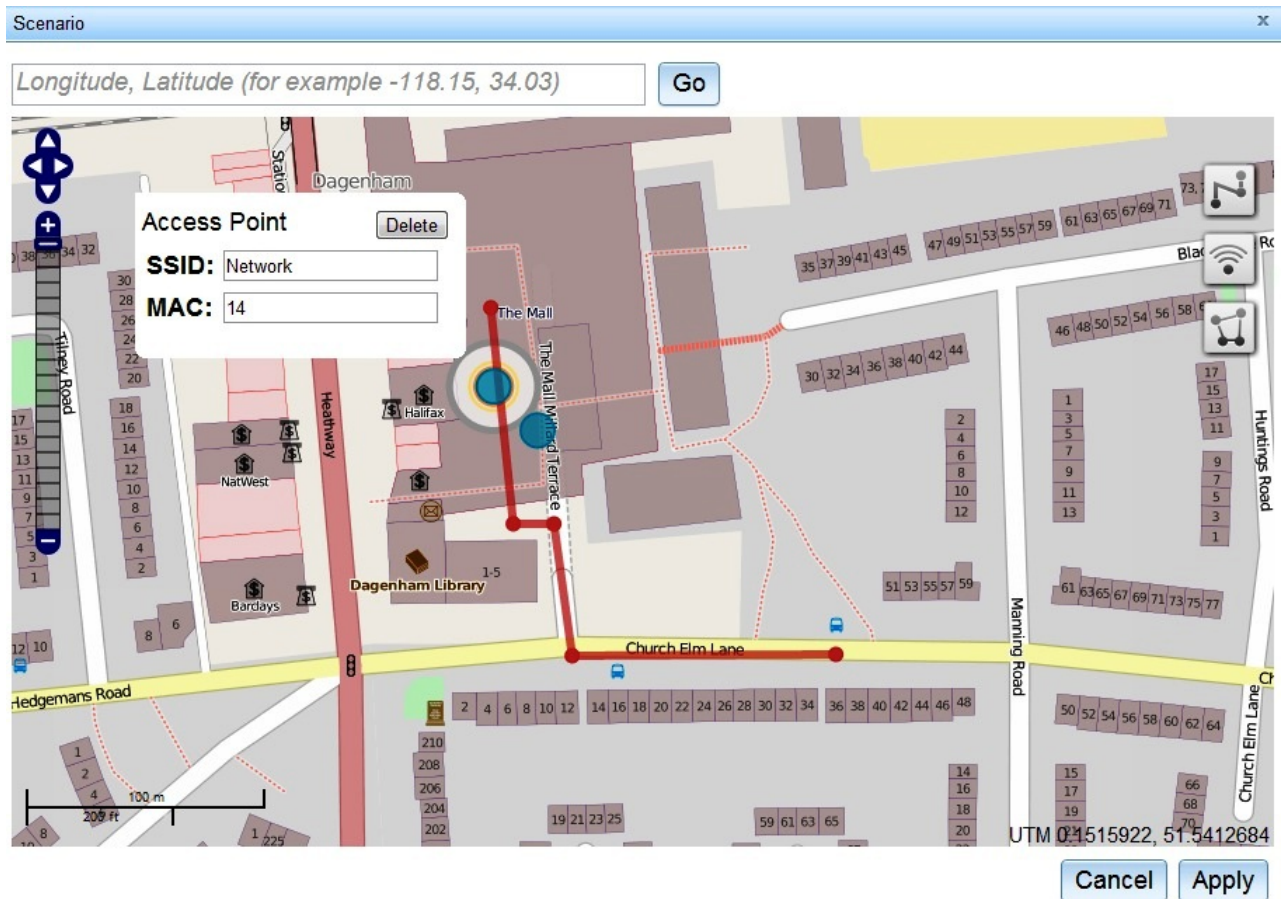
Figure 41.

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.
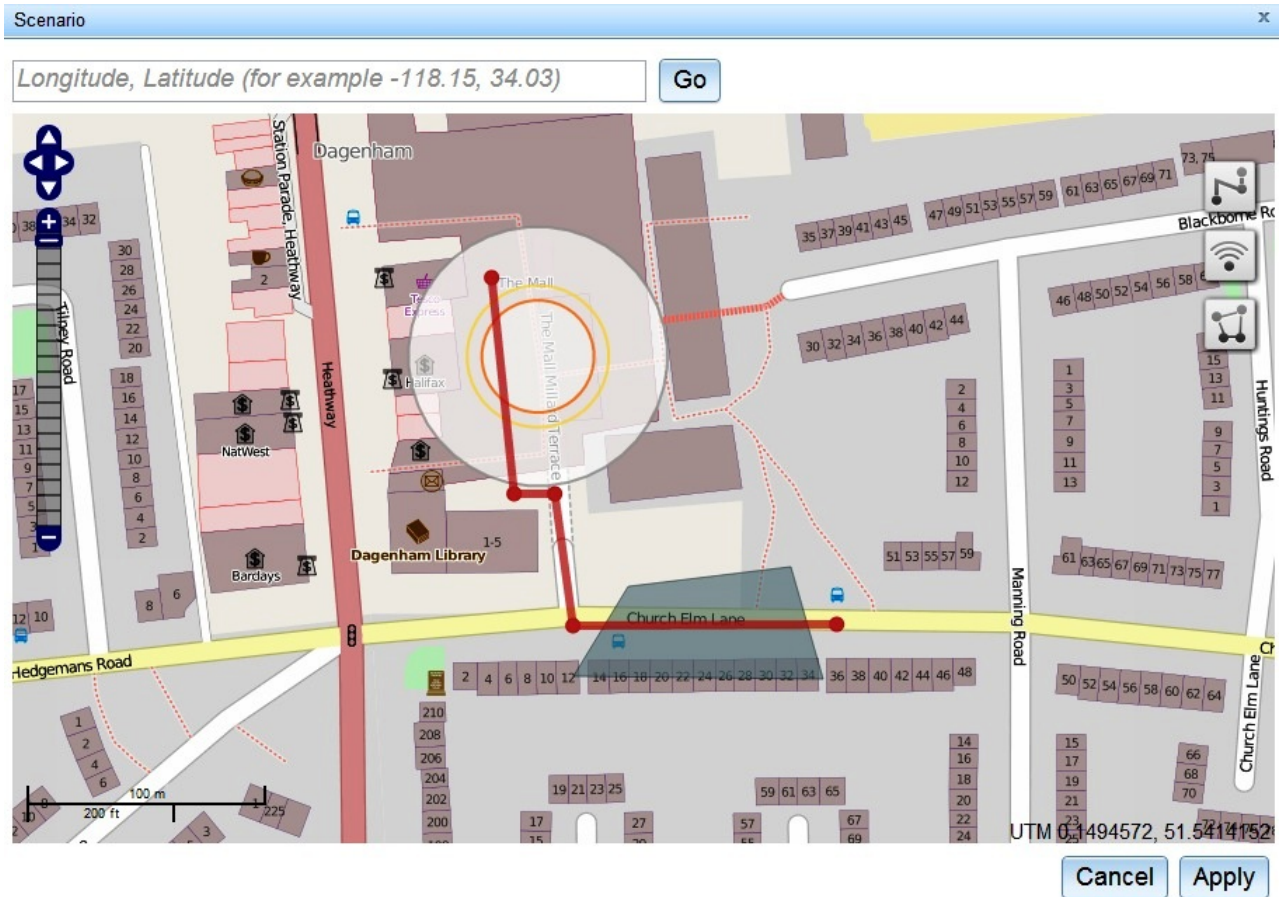
Figure 42.

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.

The second **Route Point** can be set as 11, for example

The third **Route Point** can be set as 17, for example

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 43. 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.

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, use the **WL.Client.connect()** method in common/js/main.js inside the **WlCommonInit()** method.
- 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 Worklight Logger runs on a range of client operating systems by modifying WL.Logger methods.

Enable Logger output

Enable 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 Foundation starts

You can set the logger to start when IBM Worklight Foundation starts.

You can set Worklight options to enable the logger when IBM Worklight
Foundation 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'), undefir
  //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: {lev
```

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 553 (`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.on({pretty: false});
```


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 553 is enabled (`stringify: true`).

```
var object = new Error('Boom');
WL.Logger.on({stringify: true, stacktrace: true});
WL.Logger.error(object);
```

```
WL.Logger.on({stacktrace: false});
WL.Logger.error(object);
```


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:

console.log + LogCat:

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 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 556 "Callback" on page 553,"Log message tags" on page 554, "Log package whitelist and blacklist" on page 552, and "Select log levels" on page 550.

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 44. Log output

Creating hybrid applications with IBM Worklight Application Framework

This collection of topics describes the various stages of application development when you use IBM Worklight Application Framework (Beta code).

Overview of IBM Worklight Application Framework

IBM Worklight Application Framework helps you create hybrid applications that interact with back-end services.

For example, if your company uses an existing SOAP service to manage orders, you can quickly create a mobile application that uses the back-end service and enables employees to retrieve the order information. The IBM Worklight Application Framework handles your workspace and libraries dependencies, such

as Dojo, and you can leverage IBM Worklight capabilities to deploy your project in multiple environments (iOS, Android, and others).

Your whole application is configured from a single editor, the IBM Worklight Application Framework editor. An application is defined using the following artifacts (or building blocks):

- Services: 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.
- Data Objects: 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.
- Views: Views are the representations of your application UI in the IBM Worklight Application Framework editor, and they define 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. 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.

Services   Data Objects   Views
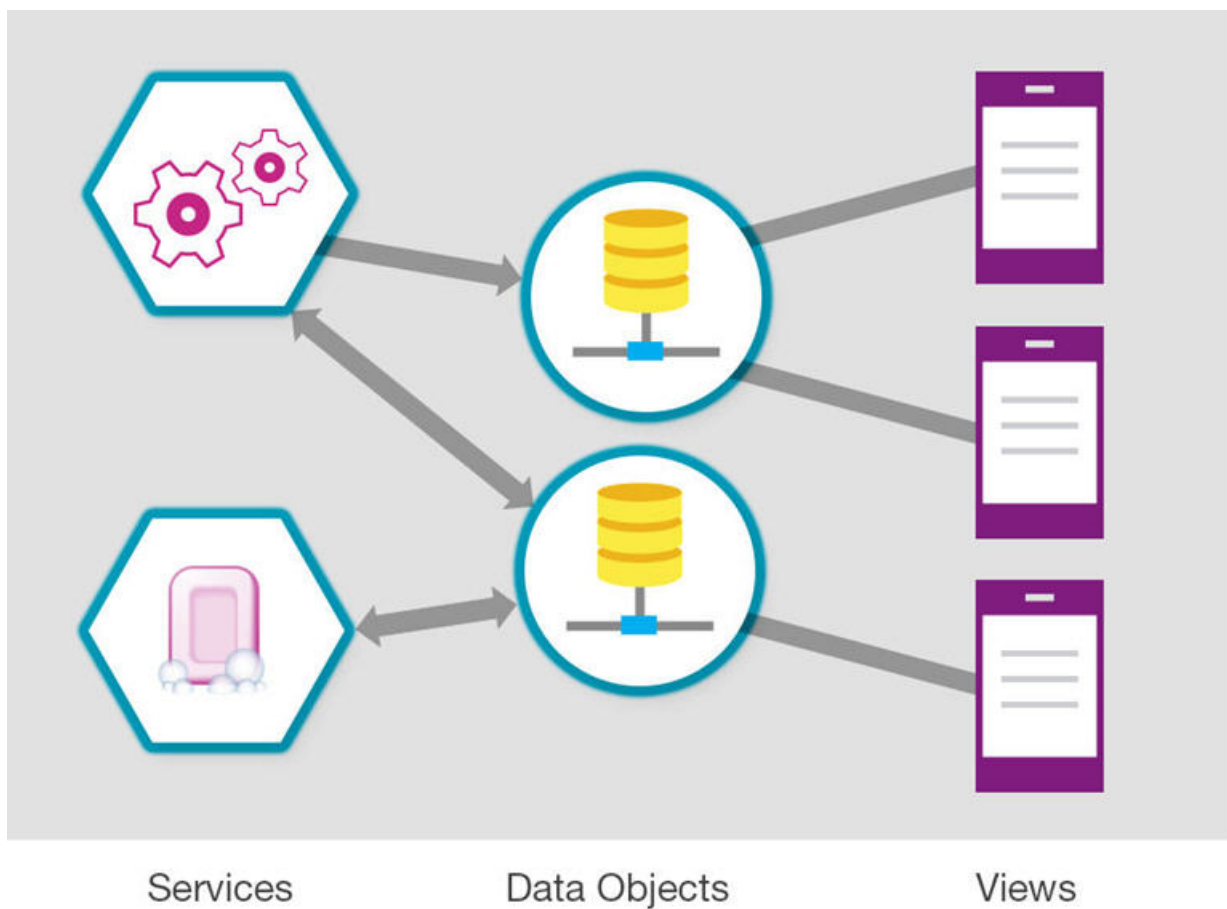
Figure 45. Interactions between services, data objects, and views in IBM Worklight Application Framework

Application resources specific to IBM Worklight Application Framework

When you choose to use IBM Worklight Application Framework to develop an application, some resources are added to the `common` folder of your app, in addition to the usual files and folders for any hybrid app:

- 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.
- The `nls` folder contains a standard Dojo i18n message bundle that is used for globalization. The `messages.js` file that is included in the `nls` folder is automatically updated according to the changes made to the UI of your application. To know more about globalization in IBM Worklight Application Framework, see "Globalization of apps created with IBM Worklight Application Framework" on page 591.

Workflow of the development process

When you start the wizard to create an IBM Worklight Foundation application, you choose to create a hybrid app, and select **Use Worklight Application Framework**. When the application 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.
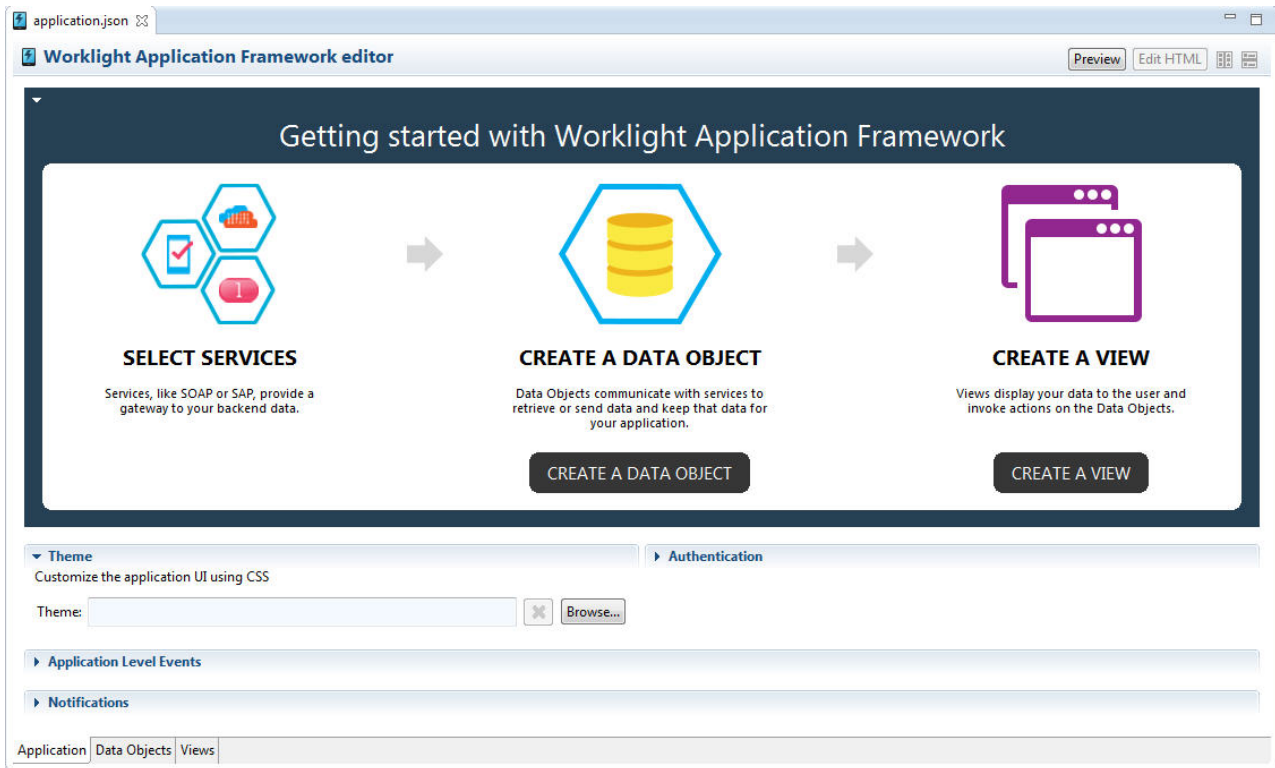
Figure 46. The **Application** *tab*

In a first stage, you start the "Add Data Object" wizard to create back-end service representations, select data objects and their attributes, and choose the operations that are required by the run time.
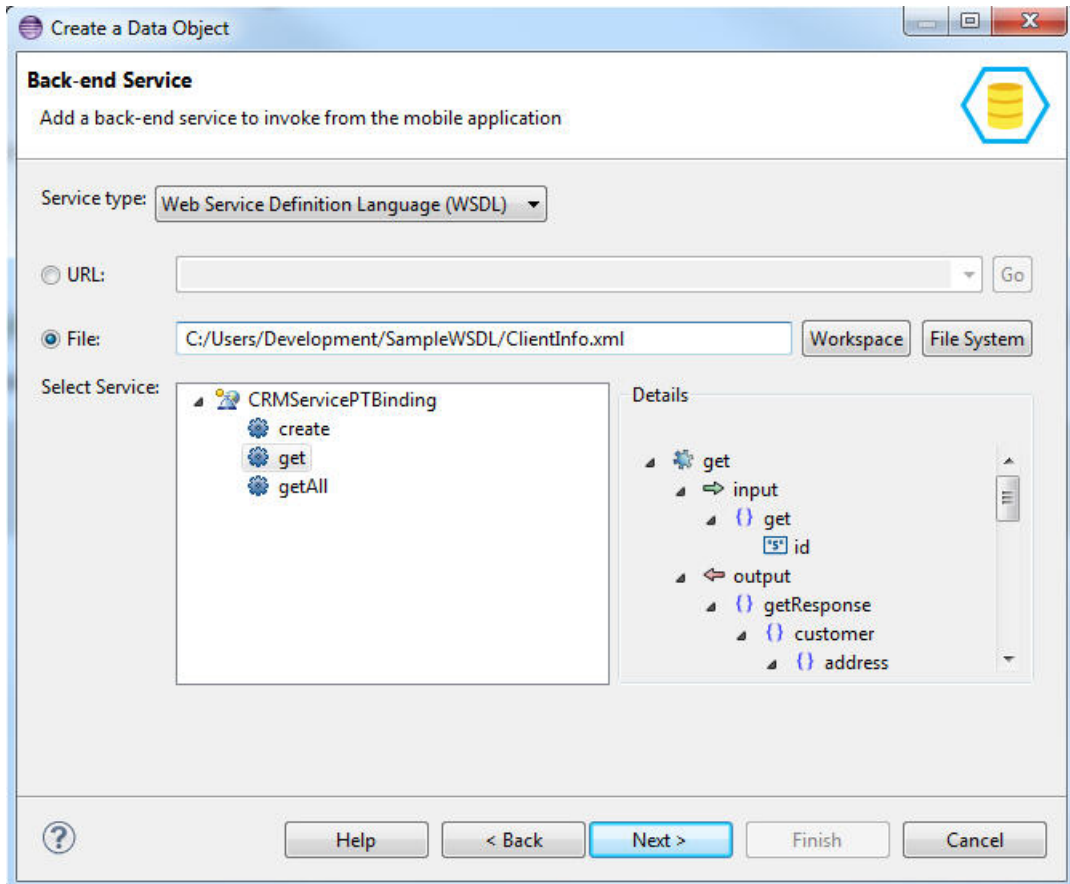
Figure 47. The "Add Data Object" wizard

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 Foundation app by following the procedure that is described in "Generating adapters with the services discovery wizard" on page 756.
- Then, you can define data objects for apps that are created with IBM Worklight Application Framework by using the "Add Data Object" wizard.

The **Data Object** tab then generates data object definitions that are 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`.

Likely to move to a specific topic about event handlers
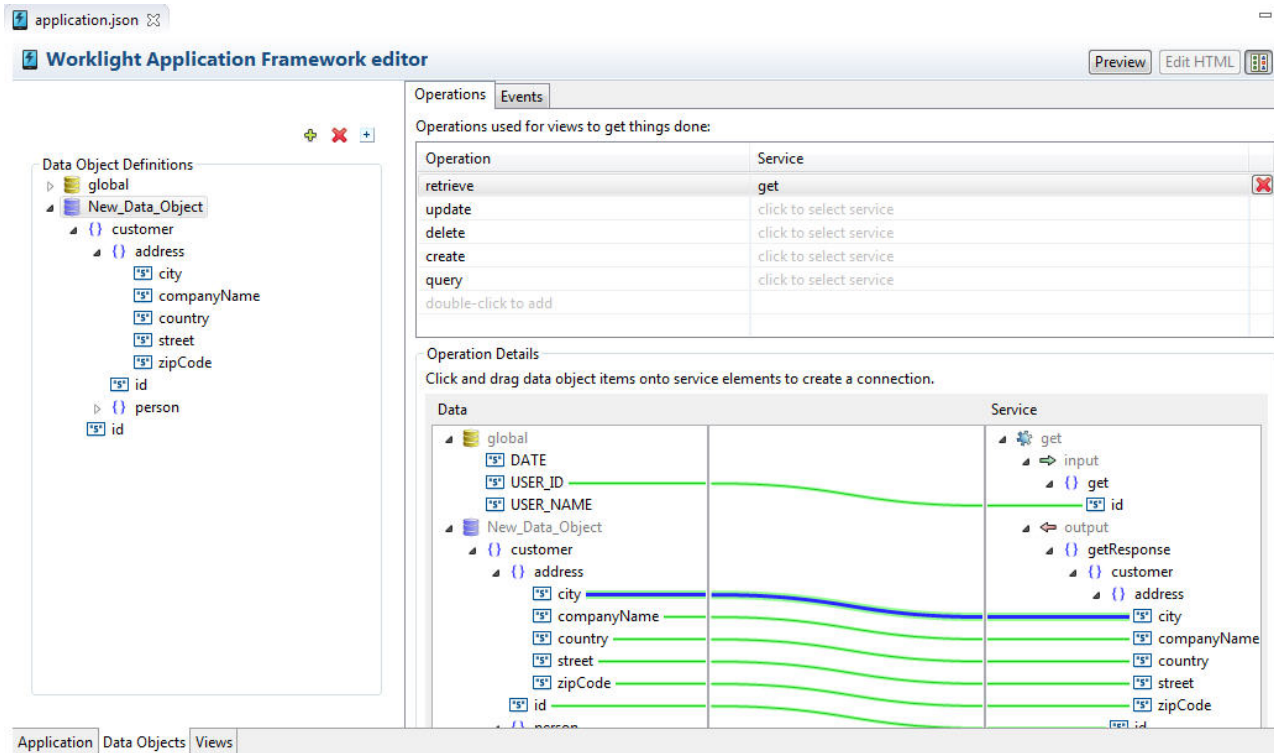
Figure 48. The **Data Objects** *tab*

You then go to the **Views** tab of the editor, and create the different UI views that are based on the data objects that you specified, and define some transitions between those views.
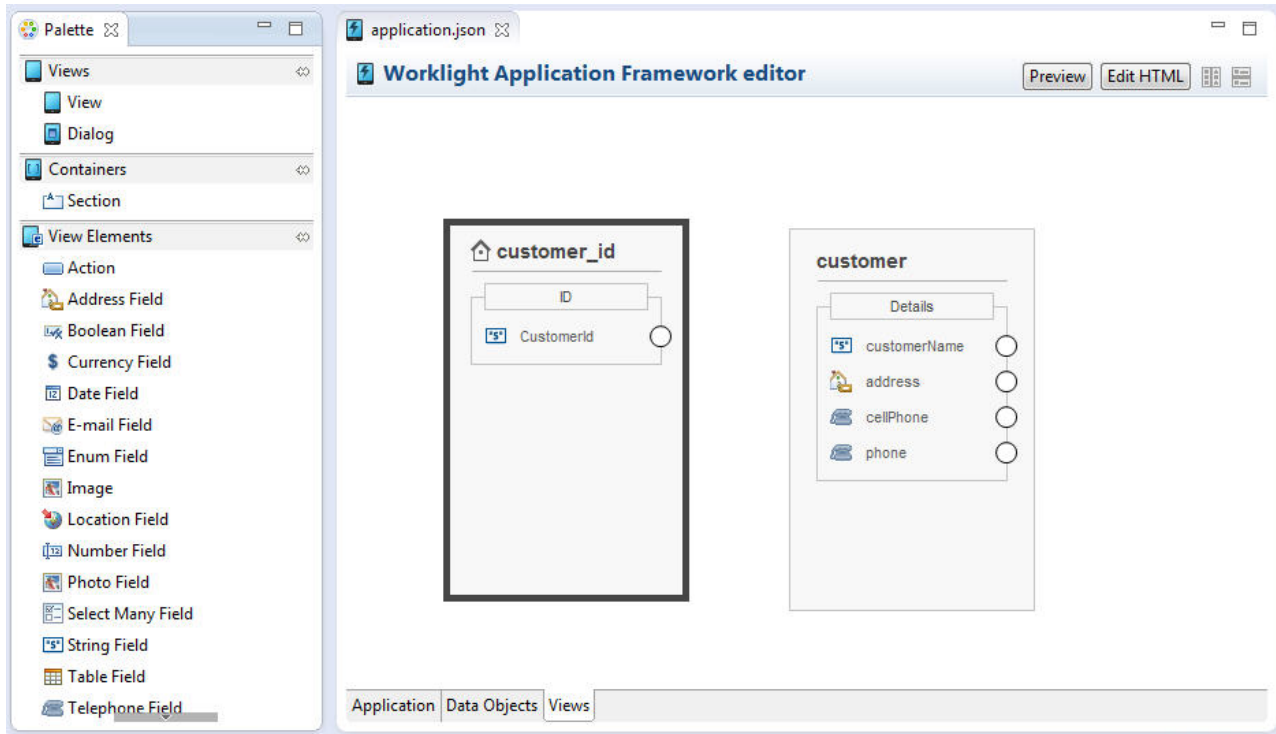
Figure 49. The **Views** *tabs*

You can then preview of your app from the IBM Worklight Application Framework editor in the Mobile Browser Simulator.
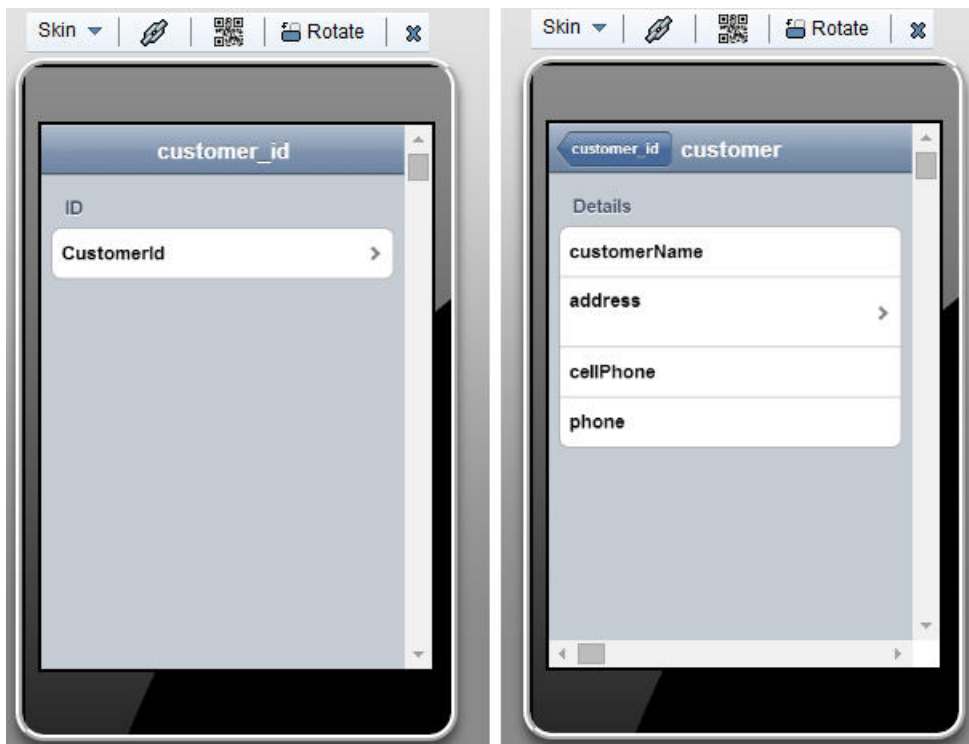
Figure 50. Preview of the application

To customize the layout of your app, or add widgets that are not related to IBM Worklight Application Framework, you can also edit the `views.html` file in Rich Page Editor.

The IBM Worklight Application Framework editor

Use the editor of IBM Worklight Application Framework (Beta code) in Worklight Studio to design your data-driven mobile application, and to define how your users interact with the app to manipulate data from existing back-end services.

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.
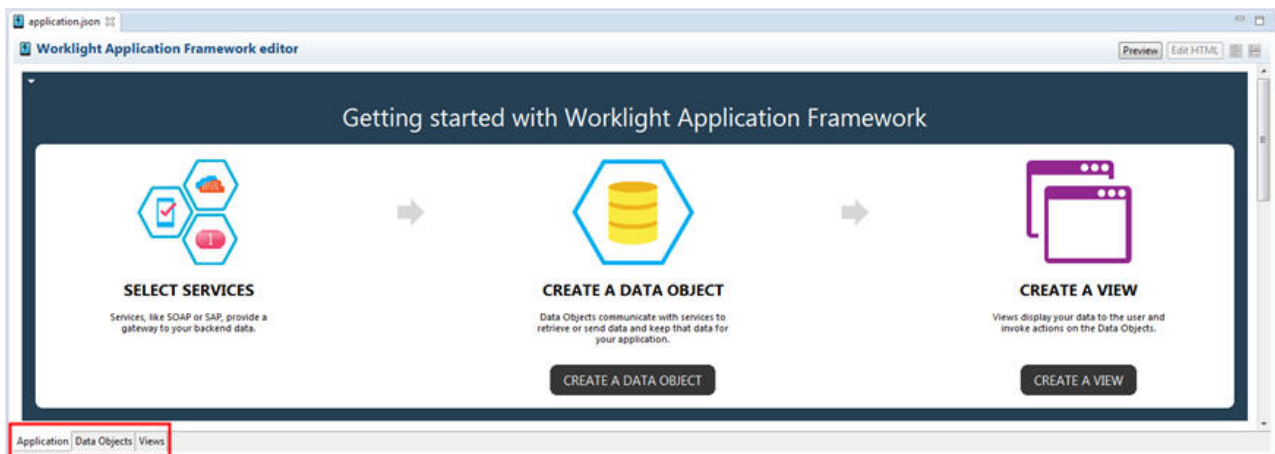
Figure 51. Multi-tabbed IBM Worklight Application Framework editor

You use the IBM Worklight Application Framework editor to edit the `application.json` and the `view.html` files that are used at run time.
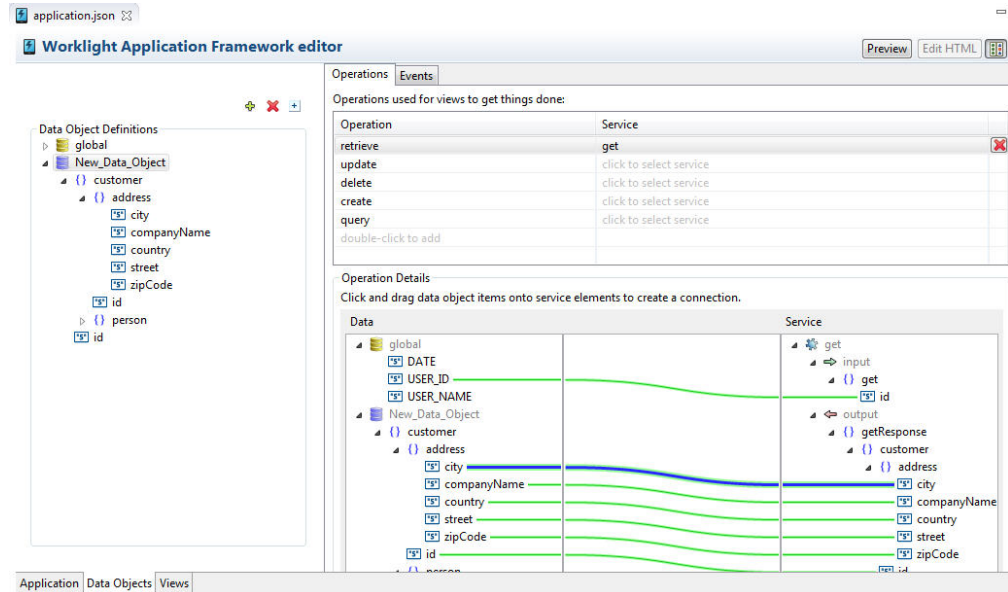
The Application tab of the IBM Worklight Application Framework editor:

Go to the **Application** tab of the IBM Worklight Application Framework editor (Beta code) to configure your application theme with CSS, configure security, or enable push notifications.

Content of the Application tab

When the project creation wizard closes after you selected **Use Worklight Application Framework**, the `application.json` file automatically opens with the default IBM Worklight Application Framework editor, and displays the **Application** tab.

From the **Getting Started** section of **Application** tab, you can open wizards that are used to discover services and add data objects to your app, and to set up views for your app.

In the **Application** tab, you can also specify a set of general configurations, such as an extra theme for your app, security parameters, event handlers, and push notifications.
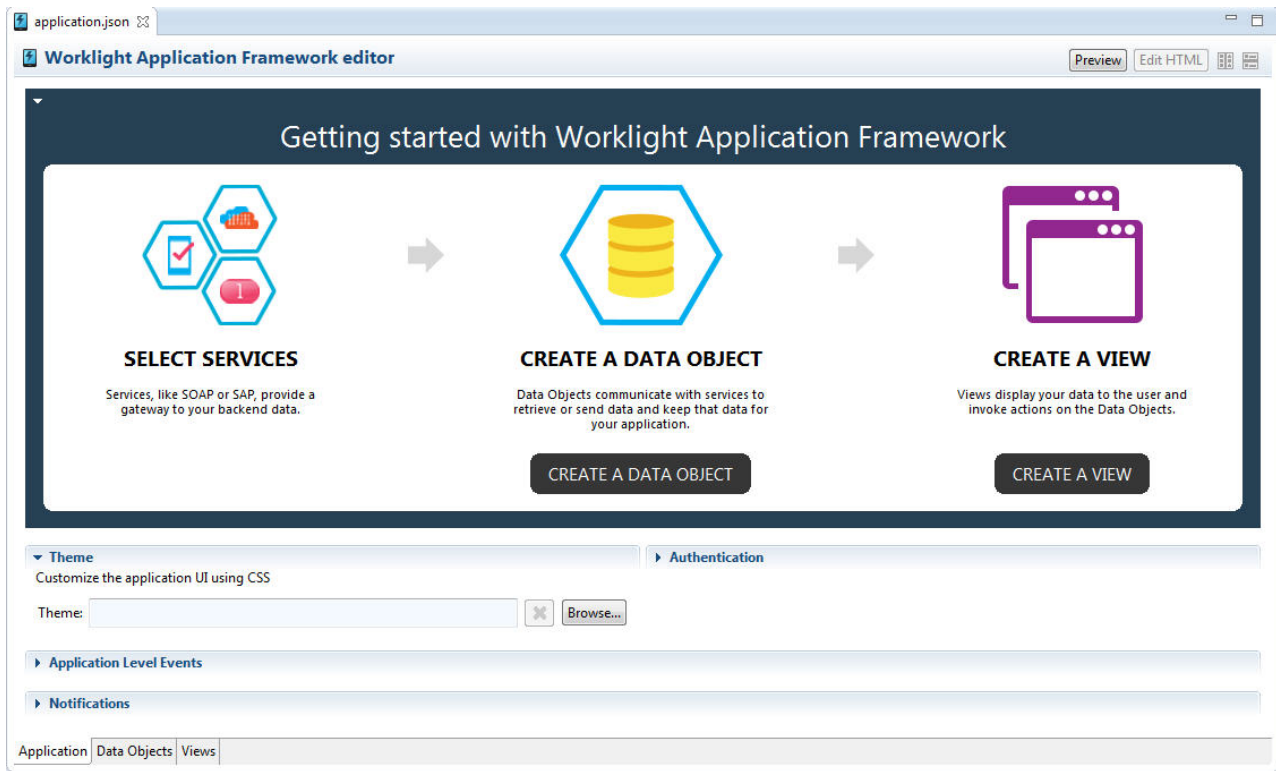
Figure 52. The **Application** *tab of the IBM Worklight Application Framework editor*

Table 95. Sections of the **Application** *tab*

Sections	Description
Getting Started with Worklight Application Framework	• If you click **Create a data object** , a wizard named **Add Data Object** 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 data objects and the services discovery process, see "Creation of data objects and usage of back-end services" on page 569. • 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 579.

Table 95. Sections of the **Application** *tab (continued)*

Sections	Description
Theme	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. If you select **User Login is required**, you can specify and review several parameters for the authentication process, such as: : • 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 872. • A re-authentication option after the end of a session. • A custom login option. For more information about how to configure a custom login view for your application, see "Setting up a custom login view" on page 586.
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.
Notifications	You can choose to use push notifications, and configure how your app works with push notifications. If you select **Enable Push Notification**: 1. Specify the **Adapter** that will send push notifications. 2. Specify an **Event Source**. 3. Specify a **View** that opens automatically when the push notification is sent. You can either select an existing view, or create one. 4. Provide a sample payload, and create the mapping between the notification content and data objects.

Creation of data objects and usage of back-end services:

You use the **Add Data Object** wizard to generate data objects that are used by the app created with IBM Worklight Application Framework (Beta code). With this wizard, you can also choose to associate your new data object with a back-end service that you specify, or to create a data object that does not use any service attributes.

About this task

The wizard provides three ways of creating data objects for your app:
- Creating a data object, discovering a new back-end service, and associating the service fields and operations with the new data object.
- Creating a data object and associating it with a service that you already added to your project.
- Creating an empty data object, without importing attributes from any service.

The services discovery process 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.

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.

To create data objects that are associated with a back-end service, you specify which type of operations you want to implement on the data, such as Create, Read, Update, Delete, or Query, and the attributes that you want to import from the service.

Procedure
1. Double-click the `application.json` file in the `common` folder of your application to open the IBM Worklight Application Framework editor.
2. Open the **Add Data Object** wizard by either:
   - Clicking **Create data object** in the **Getting Started** section of the **Application** tab.
   - Clicking the icon in the **Data Object Definitions** section of the **Data Objects** tab.

   A wizard opens.
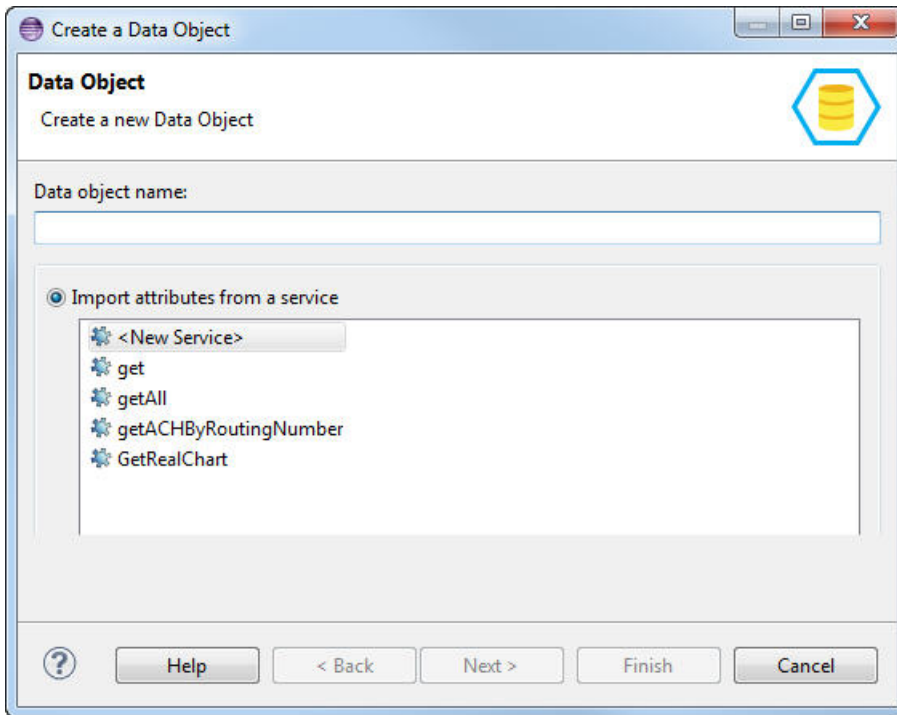
Figure 53. **Add Data Object** *wizard*

3. Enter a name for the new data object, and choose whether you want to import its attributes from a service.

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 attributes from a service**, and if you clicked the name of a service that you discovered previously from the **Import attributes from a service** list, click **Next**.

   - **Import attributes from a service**, and if you clicked **New Service** to discover a new service:

   a. Select the type of service that you want to invoke from your application.

   b. Depending on the selected type, define the service that you want to use, as described in the following sections:

      - **WSDL** service type:
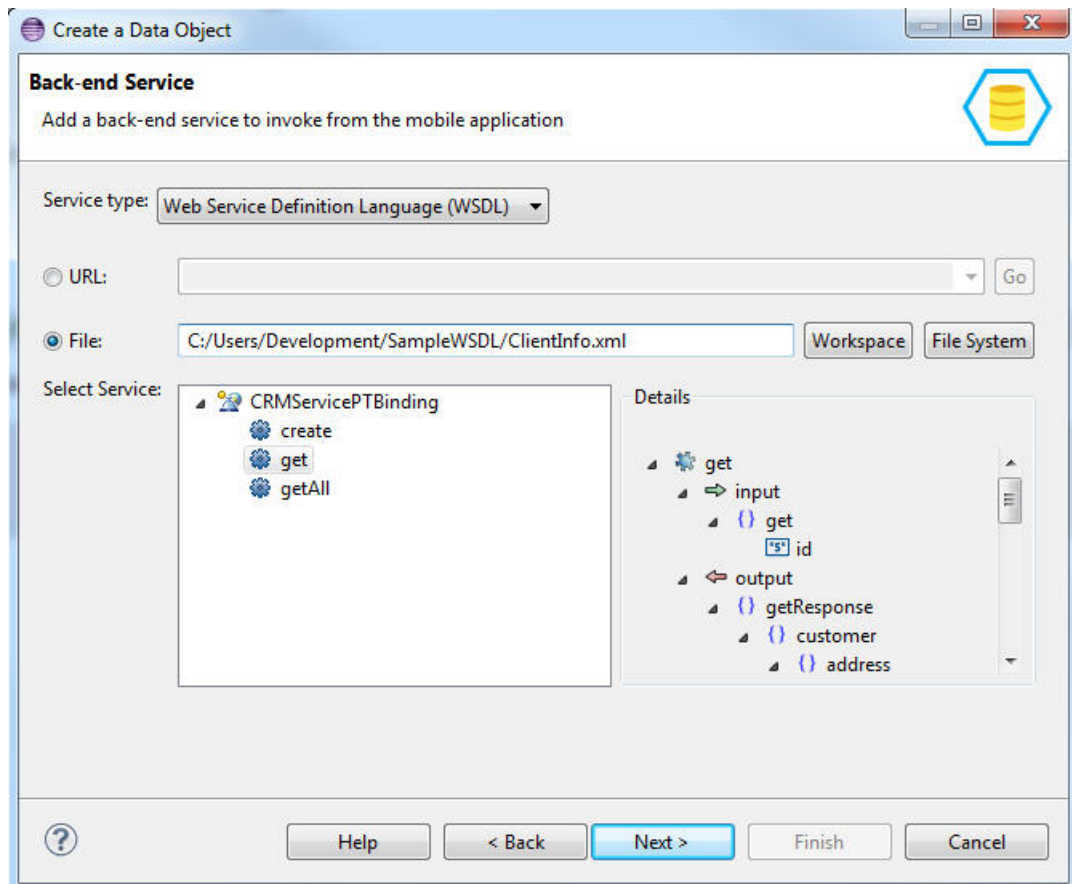
Figure 54. Adding a web service

1) 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.

2) 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 in the **Select Service** pane:

   – 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.

3) Select the data operation that you want to enable.

• **SAP Netweaver Gateway Services** service type:
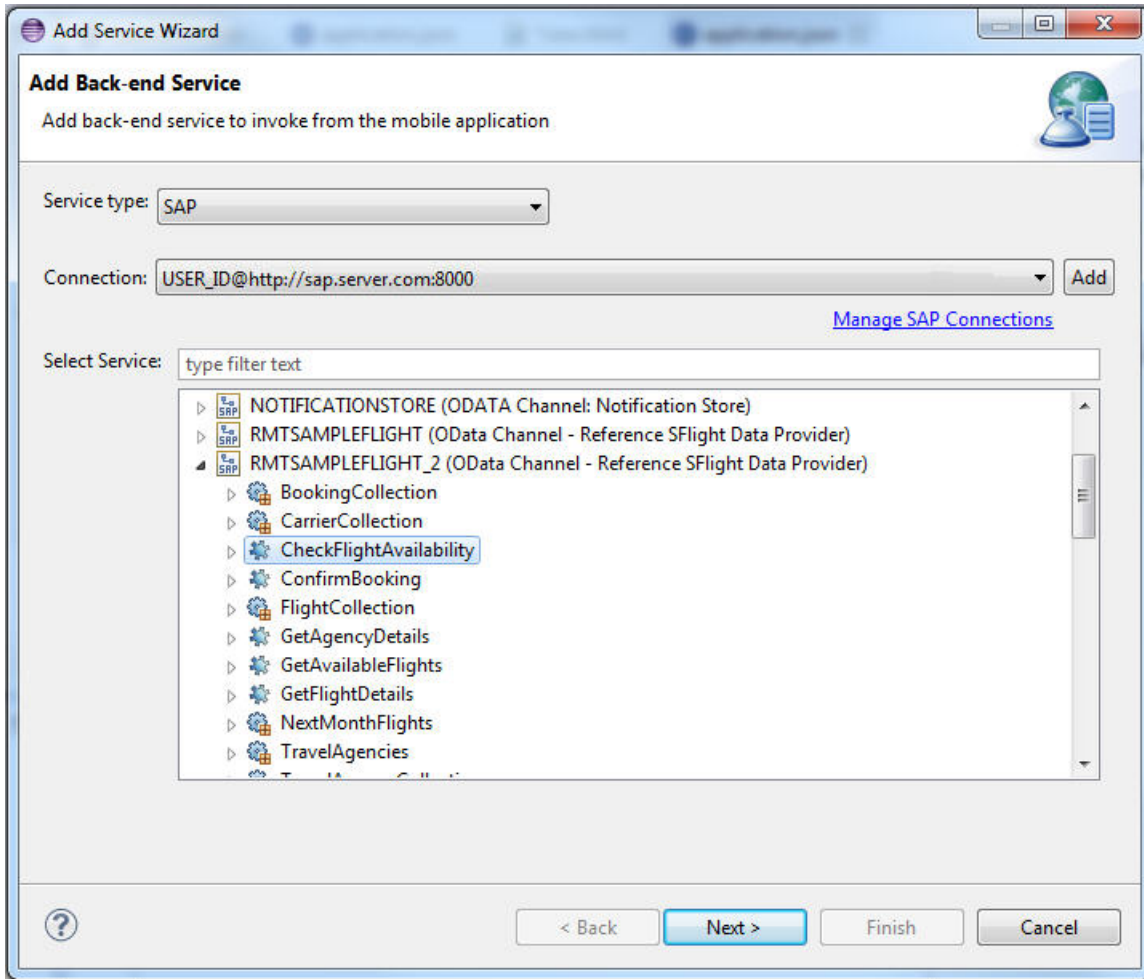
Figure 55. Adding a service exposed by SAP

1) 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.
2) 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.
3) 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.

   c. Click **Next**.

5. On the **Data Fields** page, select the **inputs and outputs** (for WSDL), or the **resource fields** (for SAP) to import.
   • Optional: You can modify certain data types, such as strings, that are specified by default in the **Data Type** column, and convert them to the type of your choice:

a. Double-click the data type that you want to modify, and choose a new type from the list.

b. Depending on the data type that you selected, you can then specify the way that the data is formatted when the data is transferred between the back-end service and your application. In the **Format** column, double-click in the cell corresponding to the data format you want to specify, and choose either a predefined format, or click **Custom** to create a new pattern.

Note: You can also change the way that the data is formatted for a data type that you modified by right-clicking the corresponding element in the **Operation Details** section of the **Data Objects** tab, and by clicking **Select format**.

6. 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.
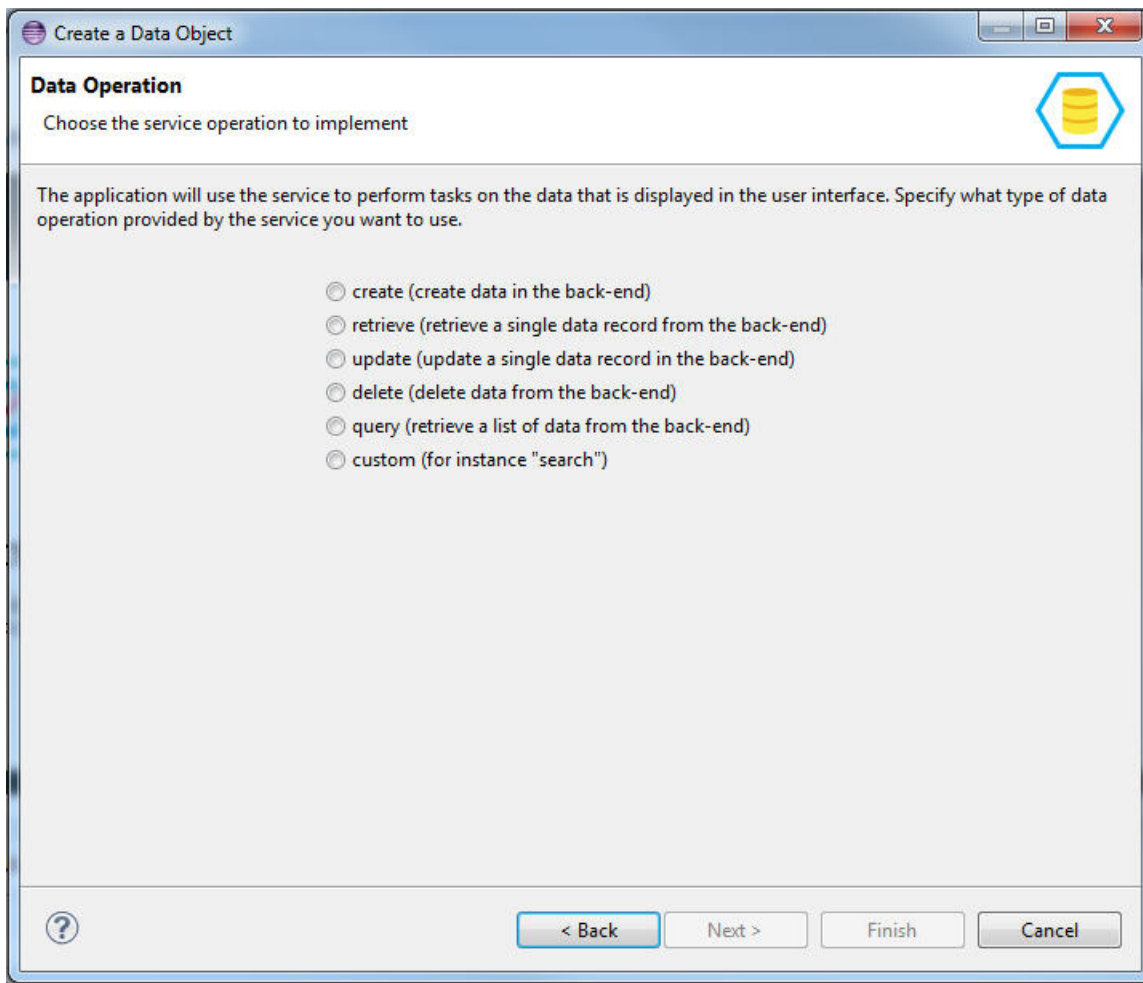
Figure 56. Data operations page for WSDL.

- For resource-oriented services, such as services exposed by an SAP Netweaver Gateway, you can define multiple operations at once.
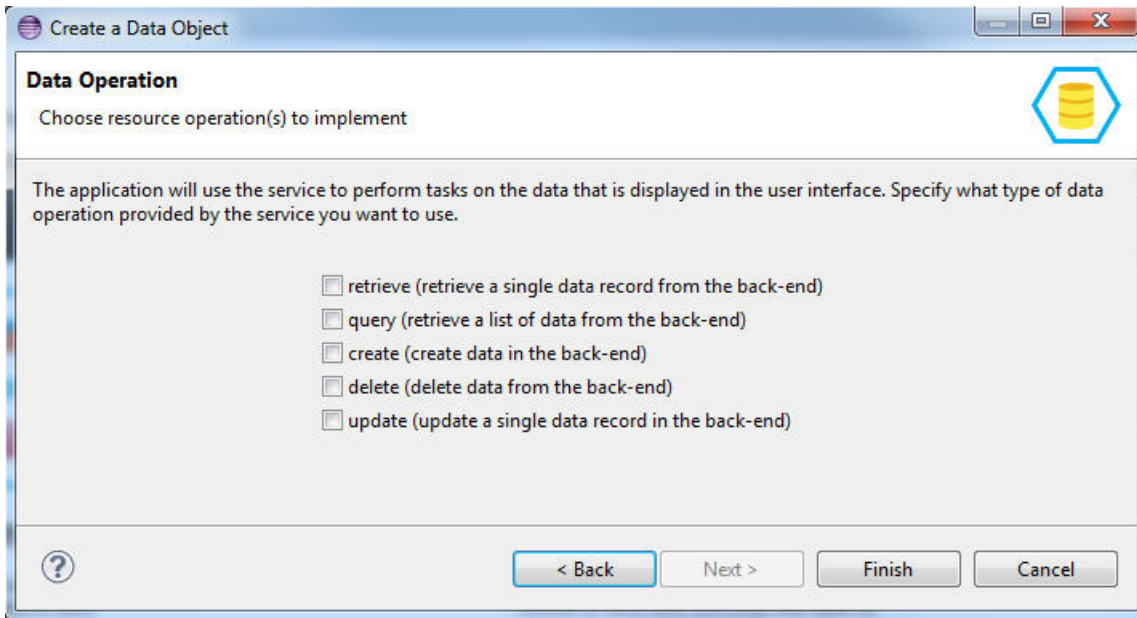
Figure 57. Data operations page for SAP

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.

If you also discovered a new service, 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 (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 "Creation of data objects and usage of back-end services" on page 569.

You then configure the data object definitions and you specify how to bind them to services for each selected data operation.

*Figure 58. The **Data Object** tab*

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 (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.
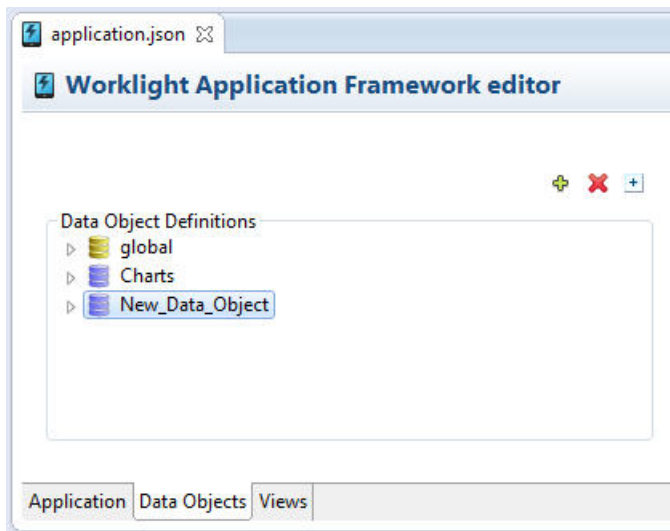
Figure 59. Data Object Definitions

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 data objects are generated from a web service where data is organized in a hierarchy, secondary generated data objects might be displayed in the **Data Object Definitions** section, in addition to the main data object entries (**DO**). Those secondary data objects are hidden by default in the **Data Object Definitions** section:

- To expand the list, and view all the data objects, click the icon ⊞ above the **Data Object Definitions** section.
- If all the data objects are displayed, and you want to hide the secondary data objects, click the icon ⊟ above the **Data Object Definitions** section.

When you expand a data object entry, you can review its attributes.

- To modify a data object or an attribute, select the data object or attribute, and click the icon ✖ to delete the selected element, or the icon ✎ to rename the selected element.
- 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, see "Creation of data objects and usage of back-end services" on page 569).
- To add an attribute to a data object, right-click the data object, and click **New**.
- When you select an attribute, you can modify its properties in the **Attribute Details** section. The data attributes properties vary depending on the data attribute type. The **Attribute Details** section includes properties such as **Required**, **Default value**, **Minimum value**, **Maximum value**, **Allowed values**, and **Maximum length**.
  - Modify an attribute property by entering a new description in the field of your choice.
  - 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.

Figure 60. The **Operations** *tab*

1. Select a data object, and an operation from the list.

   Note: To delete operations, click the icon ✖ , or to add new operations, double-click **click to select service**, 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.

> **Note:** You can change the way that the data is formatted for a data type that you previously modified by right-clicking the corresponding element in the **Operation Details** section, and by clicking **Select format**.

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 "Creation of data objects and usage of back-end services" on page 569 and in "Configuring data object definitions and customizing operation mapping" on page 575), 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.
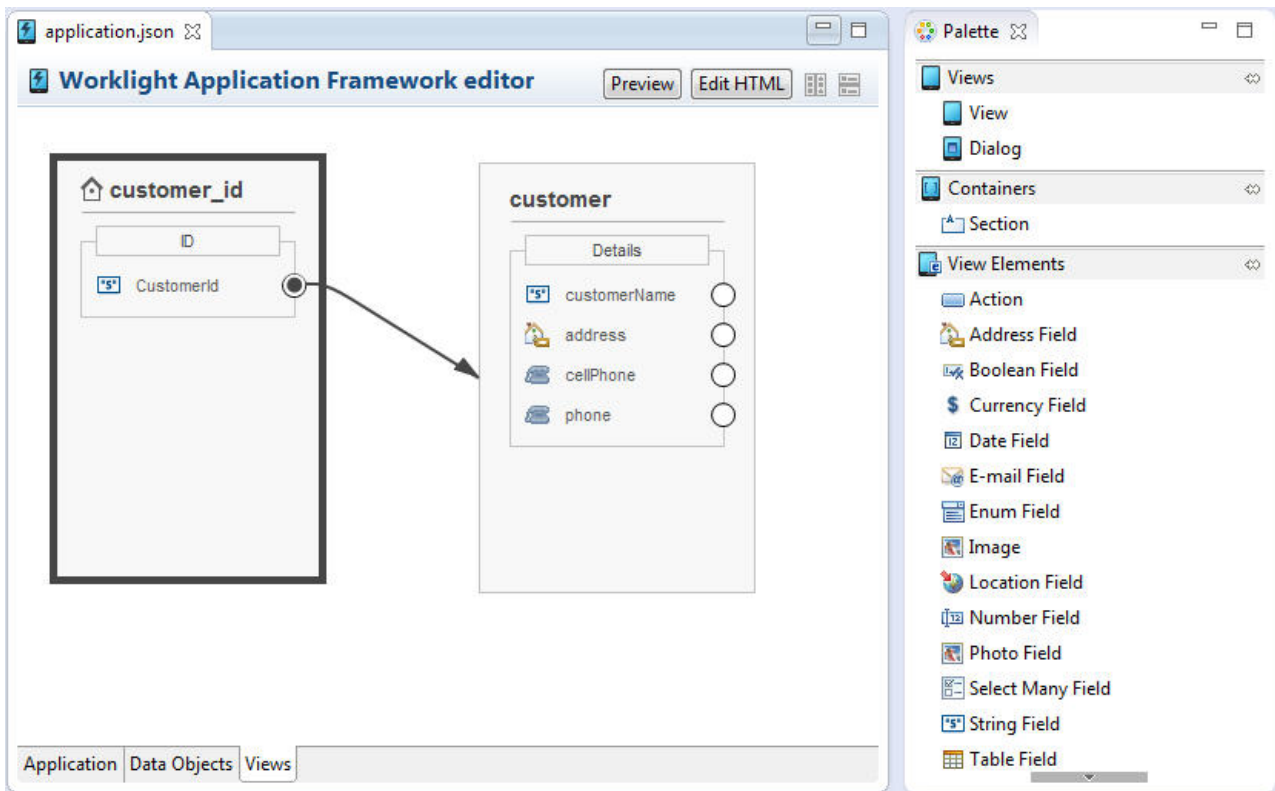
Figure 61. The **Views** tab of the IBM Worklight Application Framework editor

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.

About this task

You can either create a standard view for your app, or a dialog view (pop-up view). Two different icons are available from the **Palette** for those types of view: the **View** icon ▣ , and the **Dialog** icon ▣ .

Procedure
1. In the **Views** tab of the IBM Worklight Application Framework editor (Beta code), drag the **View** icon ▣ , or the **Dialog** icon ▣ from the **Views** section of the **Palette** to any empty area.
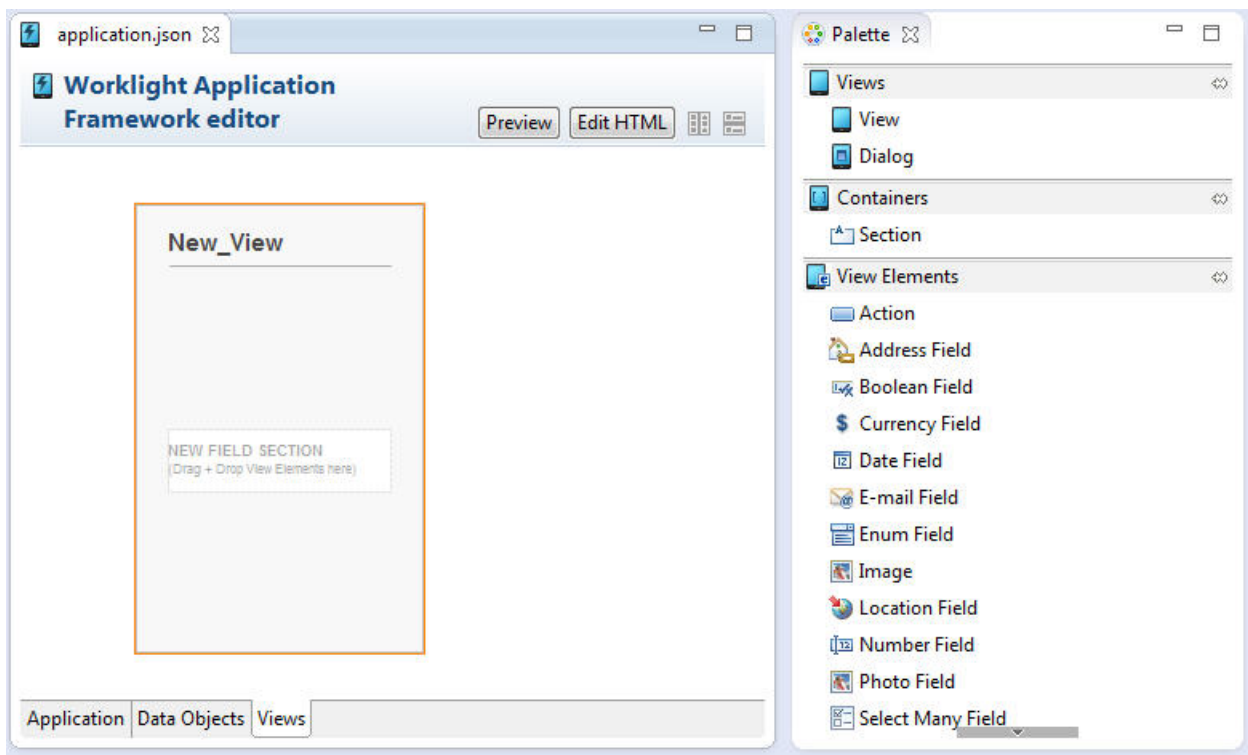
Figure 62. Create a view

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.

Note: For data objects that are generated from a web service, the **Add View** wizard lists only the main data objects (**DO**). The wizard does not display the secondary data objects that might also be generated when data is organized in a hierarchy within a web service.
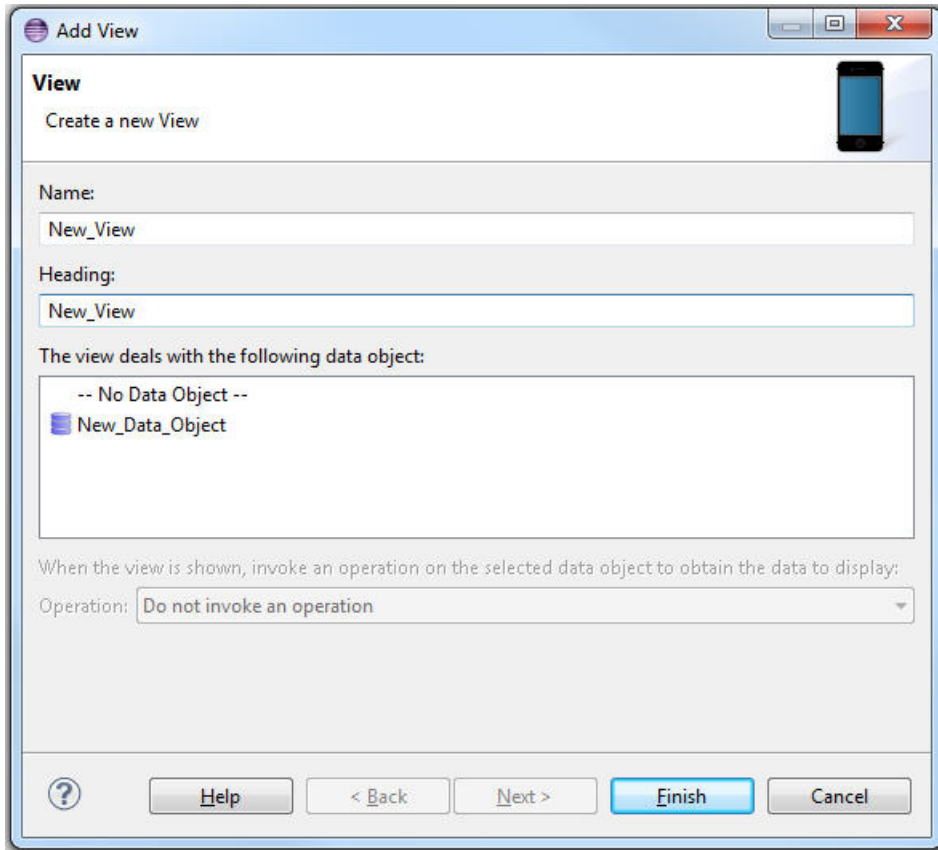
Figure 63. Defining your new view

3. Click **Next**, and select the data object attributes to display in the new view.
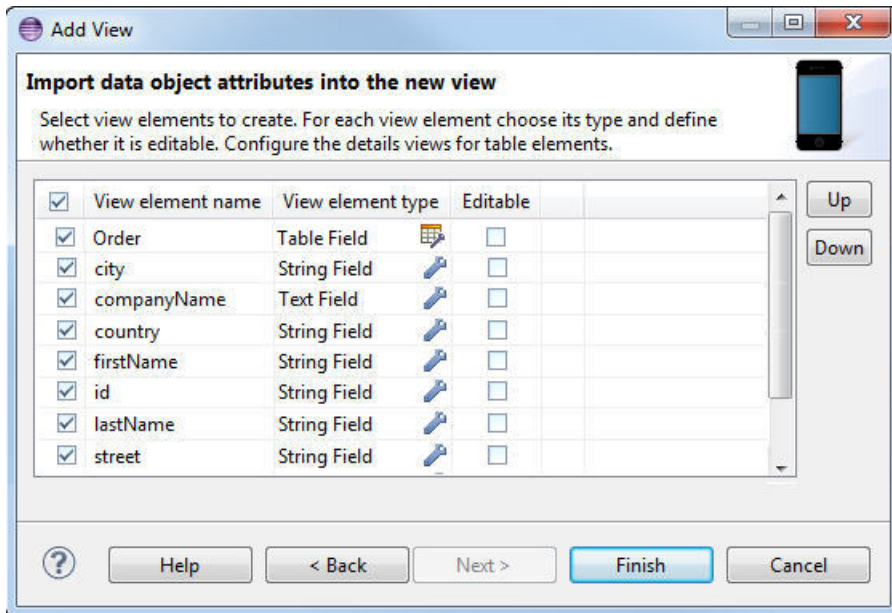
Figure 64. List of data objects attributes to import

4. Optional: You can configure view elements depending on their types:
   - For element types other than **Table Field**, you can change the type by clicking the element type of the view element that you want to modify, and by selecting a new type from the list that is displayed.
   - For **Table Field** element types, you can configure the table attributes by clicking the icon next to **Table Field**. A new page that displays the table attributes opens, and you can configure those attributes.
5. 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.
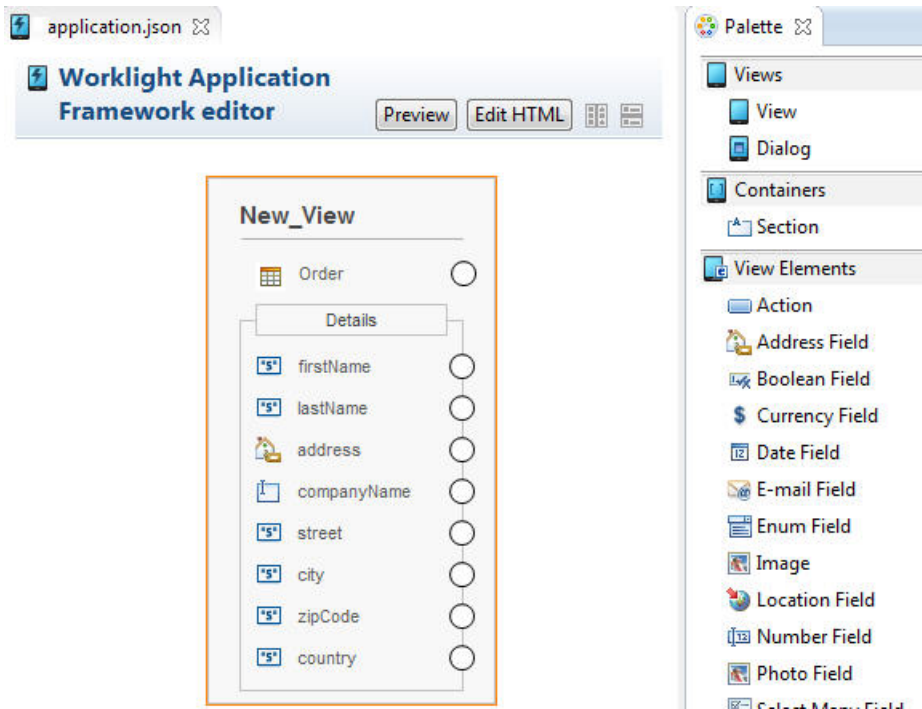
Figure 65. Result of the view creation process

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.
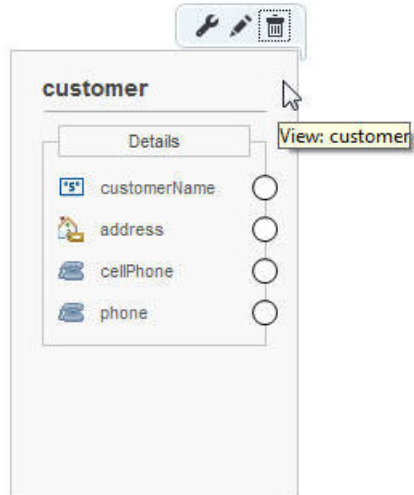
Figure 66. A view and its set of icons

To map an event handler to a view, click the icon ✖, and specify a function from your JavaScript code.

To modify the current view and configure its data objects, click the icon ✎ . In the wizard that opens, you can modify the name of the view, the heading of the view, or the type of view (standard or dialog), define operations to run, and configure the mapping between data object attributes and views.

To delete the current view, click the icon 🗑 .

Note: Initial views and dialog views are distinguishable by the icons that are displayed next to the view name: a home icon ⌂ for initial views, and the icon ▣ for dialog views. The initial view is also distinguishable by its black outlines.

Adding and modifying view elements

When you create a view, view element are added to your view according to the data object attributes you select. When a view is created, you can also use widgets from the **Palette** to insert view elements. The organization of those widgets is as follows:

- The **Section**, **Table**, **Static Text**, **Web Link** and **Image** elements must be inserted directly into a view. All the other elements must be inserted into a **Section** element that is created automatically when you drag an element to an empty space in a view.

When you hover your mouse over a section, or over a view element, a set of icons is displayed next to the section, or next to the view element.
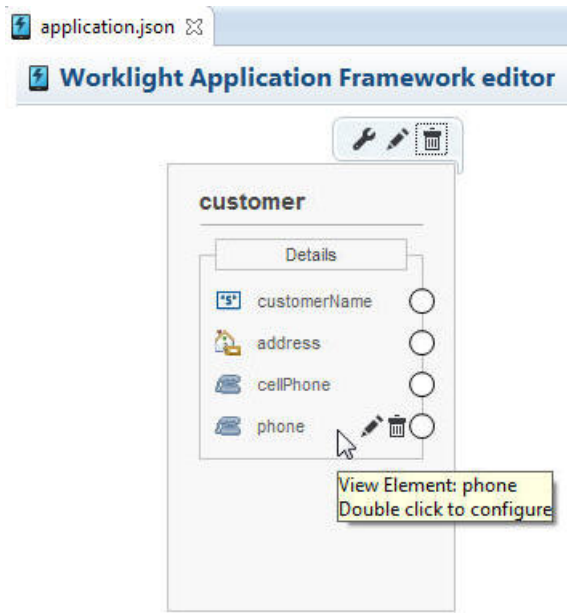
Figure 67. A view element and its set of icons

To add a section, or a view element to a view, drag the item that you want from the **Palette** to the location of your choice in the view. An **Add View Element** wizard opens. You can define the view element properties such as the name and the label. For certain view elements, you can also define the operations to run when the element is clicked and its custom event handlers.

Note: If you insert a table into a dialog view, the view for the table details must also be a dialog view.

To modify the current view element, click the icon ✏ , or double-click the view element. In the dialog box that opens, you can modify the name, the label, and other applicable view element properties. For certain view elements, you can configure the operations to run when the element is clicked and its custom event handlers.

View elements are displayed in your application in the same order as the order in which they are defined in the **Add View** wizard. To modify the order of view elements, select the view element and drag it to location of your choice in the view.

Delete a view element and its related transition, if there is one, by either selecting the view element and pressing the **Delete** key, or by clicking the icon 🗑 .

Defining an initial view

The initial view is the view that is displayed when your application starts. 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**.

Note: You cannot set a dialog view as the 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.

Note: An element in a dialog view can only trigger a transition to another dialog view, it can not trigger a transition to a standard view.

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 on 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**.

Troubleshooting the views created with IBM Worklight Application Framework:

The IBM Worklight Application Framework editor (Beta code) provides validation for some common problems. The validation process uses Eclipse infrastructure to report these problems.

To see a summary of the problems: open either the Markers or the Problems view in Eclipse. A warning icon is also displayed in the graphical editor, next to the element that contains errors.
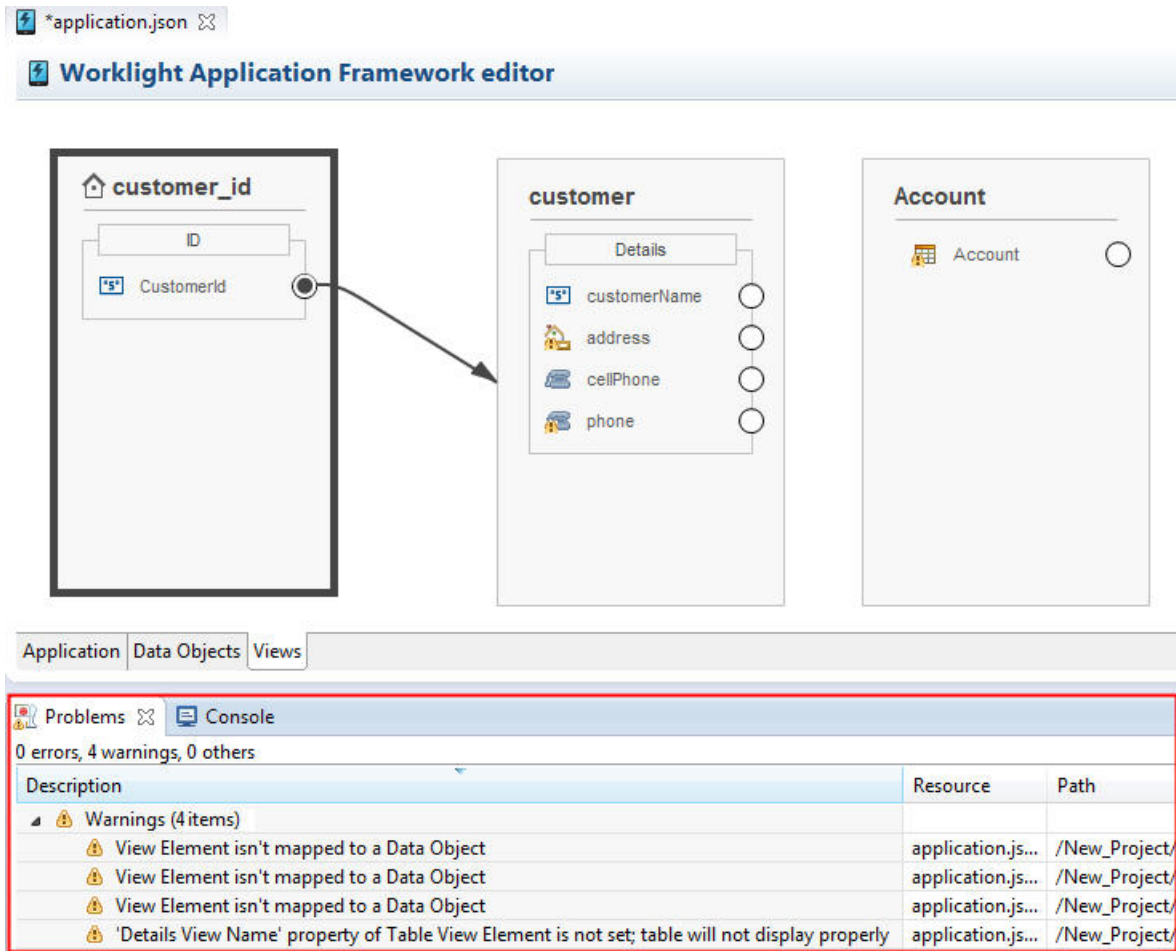
Figure 68. The Eclipse Problems view

Problems identified by the IBM Worklight Application Framework editor

Table 96. Identifying problems

Warning Message	Cause	Solution
View Element isn't mapped to a Data Object	A mappable view element is not associated to any data object.	Click the pencil icon 🖊 on top of the view. In the section **Specify how the view elements are bound to the data object instances** of the wizard that opens, map the view element to a data object.
'Details View Name' property of Table View Element is not set; table will not display properly	A table element does not have a **Details View** setting.	Click the pencil icon 🖊 next to the table element in your view, and set a value for the **Details View Name** field.

Setting up a custom login view:

You can create a custom view, and define this new view as a login view instead of the default login view that is provided by IBM Worklight Application Framework (Beta code) at run time.

About this task

You create a data object, and you associate this data object with a login view that you also create. You then add JavaScript methods to send the user credentials to the server, and to handle the custom login view on authentication response (success or failure).

Procedure
1. In the **Data Objects** tab of the IBM Worklight Application Framework editor:
   a. Create a simple data object, which is not connected to any service, for the authentication process.
   b. Add attributes to the data object that you created in step 1a for a user name, a password, and an error message that is sent by the server in case of an error.

   For more information about how to create and configure data objects, see "Creation of data objects and usage of back-end services" on page 569 and "Configuring data object definitions and customizing operation mapping" on page 575.
2. In the **Views** tab of the IBM Worklight Application Framework editor:
   a. Create a view for login, and associate it with the data object that you created in step 1a.
   b. In your new view, add a view element (for example, a **Button**) to trigger the sending of the credentials to the server.
   c. Edit the trigger element that you created previously, and bind a JavaScript function to the onTap event so that the user credentials are sent to the server when the view element is clicked. The following example illustrates the recommended structure for the login trigger:

   ```
   doOnLogin: function(ctx) {
       var app = ctx.getApp();
       var view = app.getCurrentView();
       var viewModel = view.getDataObject('loginObj');
       var userName = viewModel.getValue()['user'].value;
       var pass = viewModel.getValue()['password'].value;
       app.submitAuthData(userName,pass);
   }
   ```

   For more information about how to create and configure views, see "Adding a view to your application" on page 579 and "Configuring the views of your application" on page 582.
3. In the **Application** tab of the IBM Worklight Application Framework editor:
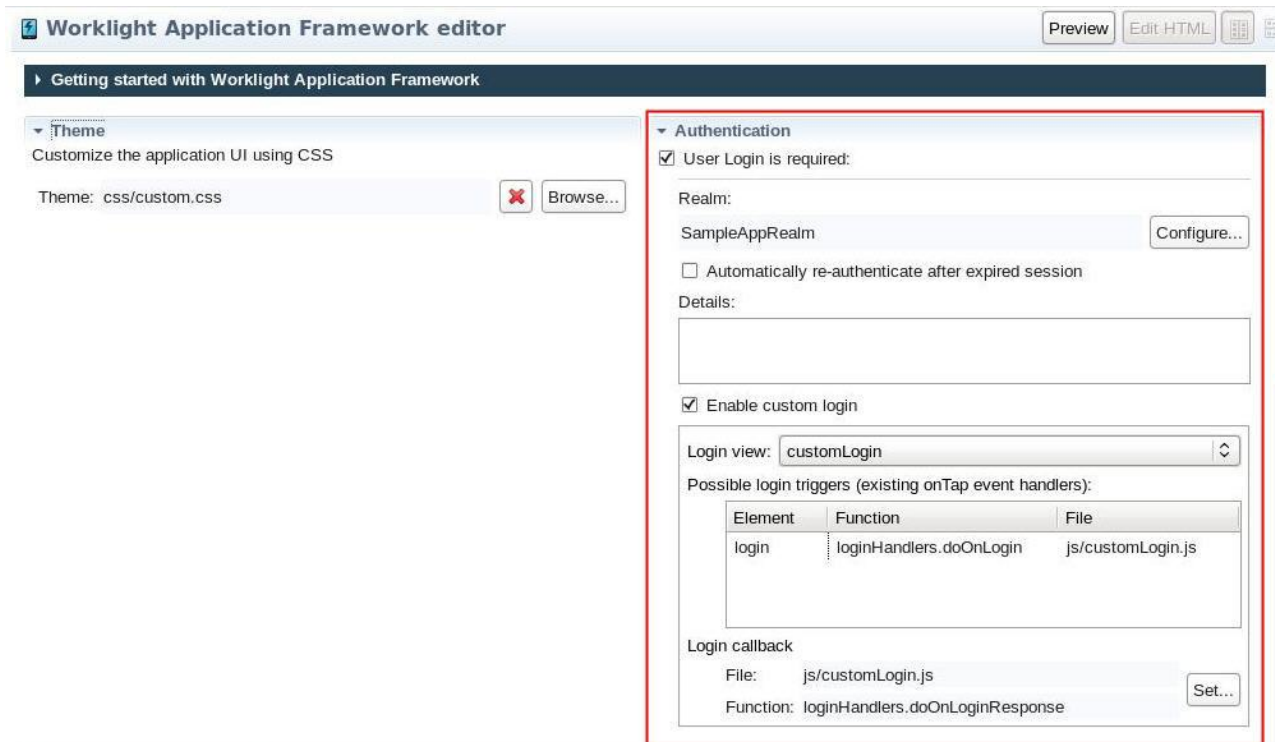
Figure 69. The Authentication section of the IBM Worklight Application Framework editor

a. Select **User Login is required** in the **Authentication** section of the tab, and choose the login parameters that are based on the available realms.

b. Select **Enable custom login**.

c. In **Login view**, select the view that you created in step 2.

d. Check that the event trigger that you set up in step 2c is displayed in the **Possible login triggers** table.

e. In **Login callback**, click **Set** to specify the custom JavaScript method to be called at the reception of the success or failure response. The following example illustrates the recommended structure of the onLoginResponse method:

```
doOnLoginResponse: function(ctx, success, errMessage){ // The server response has a context, a
    if(!success){ // If boolean indicating success is failing, then you should show an error
        this.showError(errMessage);
    } else {
        // Complete the logic on your application login success
    }
}
```

Results

- If all the settings for your custom login view are correct, you can save the application.json file.

- If your configuration is incorrect, an error message that indicates the cause of the error is displayed at the top of the IBM Worklight Application Framework editor. You cannot save the application.json file when the custom login configuration is incorrect.

Previewing your app from the IBM Worklight Application Framework editor:

You can preview an application created with IBM Worklight Application Framework (Beta code) from any tab of the editor, and at any stage of the development process.

Procedure
1. Click **Preview** in the upper-right corner of the editor.

Figure 70. **Preview** *in the IBM Worklight Application Framework editor*

The Worklight Development Server starts (if not active yet).
2. Optional: If a dialog box opens and notifies you of the automatic deployment of adapters, click **OK**.

This dialog box opens only if your application uses Worklight adapters to access back-end services, and if these adapters are not yet deployed.

Note: Even if you choose not to display this message again by selecting **Never show this dialog again**, the required adapters are always deployed automatically.

Results
- When the application is ready for preview, and if you added environments to your app, the Mobile Browser Simulator opens and displays all your current environments.
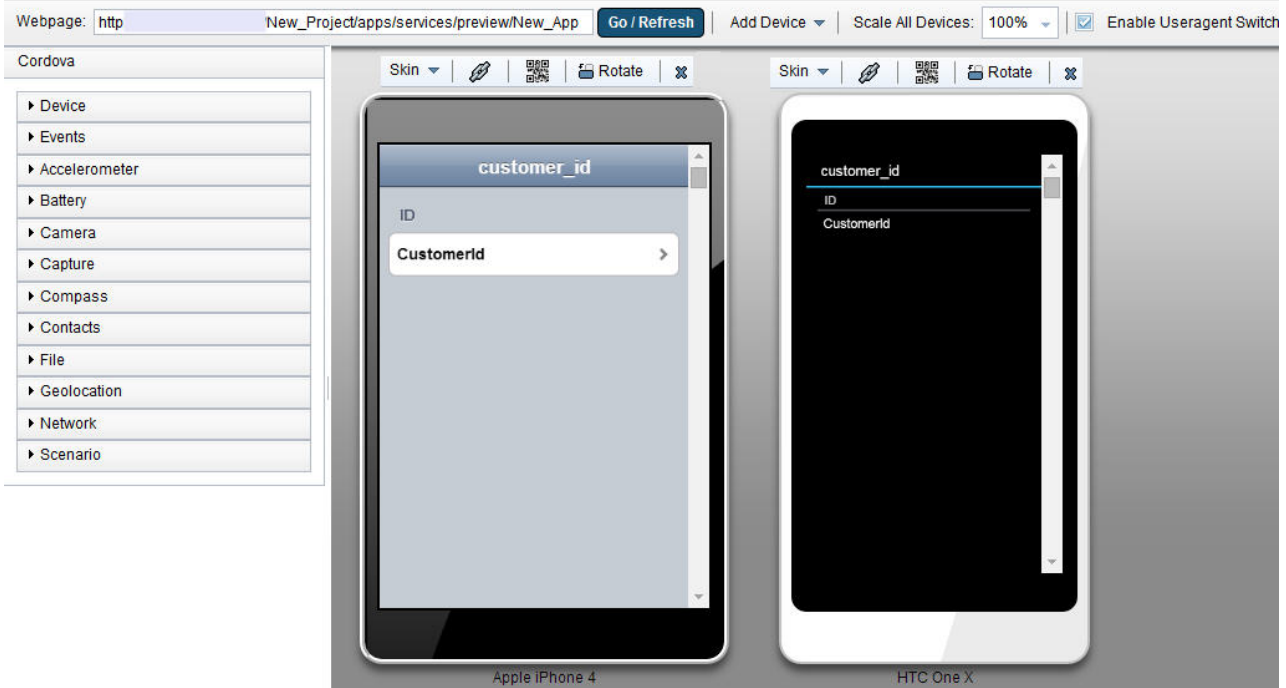
Figure 71. Environments previewed in the Mobile Browser Simulator

> To know more about how to use the Mobile Browser Simulator, see "Previewing your Worklight applications" on page 531.

- If no Worklight environment is configured for your application, a standard preview page opens (instead of the Mobile Browser Simulator).

Figure 72. Standard preview page with no specific environment

Globalization of apps created with IBM Worklight Application Framework

Every application that is created with IBM Worklight Application Framework can now be globalized via a standard Dojo i18n message bundle that is included in each application folder.

All the view titles, element labels, and element hints come from a standard Dojo format message bundle. When the application starts, the run time loads the appropriate locale-based bundle.

The `nls` folder and the `messages.js` file

The globalization bundle is under the `common/nls` folder of each application, with a `messages.js` file.
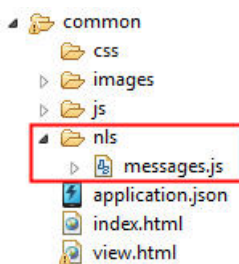
The `messages.js` file is automatically maintained by IBM Worklight Application Framework. For example, changes to view titles and to view element labels are automatically reproduced in the message bundle.

The keys in the bundle are as follows:

- View title: *name_of_the_view*
- View element label: *name_of_container_view.name_of_view_element*. For example, if an element named *x* is in a view that is named *H*, the key for the view element label is *H.x*.
- View element hint: *name_of_container_view.name_of_view_element.hint*. For example, if an element named *x* is in a view that is named *H*, the key for the view element hint is *H.x.hint*.

Web and native code in iPhone, iPad, and Android

Using IBM Worklight Foundation, 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 between native and web views

In iOS and Android applications, natively developed pages can be invoked from your web-based pages and can then return control to the web view and vice versa. 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 to a native view from a web view

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

Authenticating in native and retrieving user information in web view

The Worklight JavaScript API provides the following methods for retrieving
authenticated user information, such as the login name, display name, or any other
custom user identity property.
- WL.Client.updateUserInfo()
- WL.Client.getLoginName()
- WL.Client.getUserInfo()
- WL.Client.getUserName()

Note: The preceding methods are available in the JavaScript API only. If your
application implements authentication in the native layer and only then moves to
the web layer, you must invoke WL.Client.updateUserInfo() methods in order to
retrieve authenticated user information from the Worklight server.

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 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
}
-(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 Worklight client framework makes the data available to the `Activity` in a `Bundle`.

Example

Sending data from web view to the native Activity:

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

Receiving the data in the native Activity:

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

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

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

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

Related information:

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

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

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

Returning control to the web view from a Java page
To switch back to the web view, follow these instructions

Before you begin

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

Procedure

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

Example

Passing data and control to the web view:

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

Receiving the data in the web view:

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

Related information:

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

Animating the transitions from and to a Java page

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

Procedure

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

Example

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

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

Related information:

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

Guidelines for using native code in Worklight projects

Follow these guidelines to keep your project's native code intact during a build.

As a rule,Worklight builds are non-destructive with respect to files in the **native** folder. The exception is the target **web** folder (meaning **/www**, or **assets/www**), which is overwritten during each build with the optimized web resources from **common** and environment-specific folders. Keep to the following rules to ensure that your native code remains intact during a build:

- If you have edited native code in either Android Studio or Xcode, ensure that your changes are under the app's `native` folder before the next build in Worklight Studio.
- You can safely place and edit non-web assets in the `native` folder. No duplication or copying of files under `native` to other locations occurs during a build.

- Additionally, you can place a limited number of assets (for example, icons) under the `nativeResources` folder that is then copied into the `native` folder during a build. However, make this the exception, rather than the rule.
- Never edit web assets directly in the `native` folder. Edit web assets in either the **common** or environment-specific folders, as appropriate. These folders are merged and optimized during a build and placed in the proper web target folder.
- Do not manually delete the `native` folder as it may contain critical custom code that you have implemented. If you must delete the folder, first ensure that you have backed up the code.
- Follow the guidelines for using source control systems with Worklight projects: See "Integrating with source control systems" on page 418.
- Use the `nativeResources` folder to hold static resources only, such as images.

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 Foundation 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 "Worklight security framework" on page 860.

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

IBM Worklight Foundation 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 Foundation 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

IBM Worklight Foundation 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 Foundation 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 Worklight configuration properties, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182. For information about how to specify Worklight configuration properties, see "Configuration of Worklight applications on the server" on page 1171.

About this task

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:
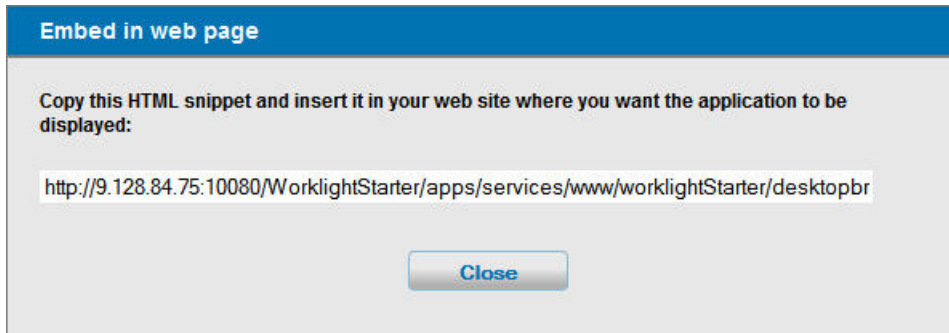
Figure 73. Embedding a widget in a predefined web page

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;"> <
```

Configuring and customizing direct update

Direct update is the direct delivery of 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. In IBM Worklight Foundation, this option is available for iPhone, iPad, Windows Phone 8, and Android apps.

For an introduction to direct updates of app versions to mobile devices, see "Direct updates of app versions to mobile devices." A direct updates mechanism is available for desktop apps. For more information, see "Direct updates of app versions to desktop apps" on page 603.

The Worklight Server can push data at the rate of 250 MB per second. For example, if an application is 5 MB in size, assuming that the network bandwidth is not a bottleneck, the Worklight Server can serve 50 direct updates per second, and a Worklight Server cluster of four servers can serve 200 direct updates per second.

To serve direct updates at higher rates, consider using a CDN (content delivery network) instead of the Worklight Server.

Direct updates of app versions to mobile devices

The Worklight Server can directly push updated web resources to deployed applications.

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.

Direct update works only if the client-side artifacts of the application are built with the same version of Worklight Studio with which they were originally built when deployed to Worklight Server. See "Upgrading Worklight Studio in the Consumer or Enterprise Editions to Worklight Studio V6.2.0" on page 347 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 the following figures show the default method of implementing Direct update. You can customize the direct update process and interface of apps developed in Worklight Studio, V6.2.0 and later. For more information, see "Customizing the direct update interface and process" on page 608.
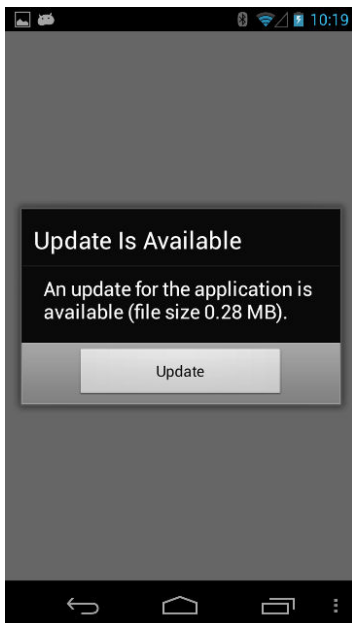
Figure 74. Update notice from Android
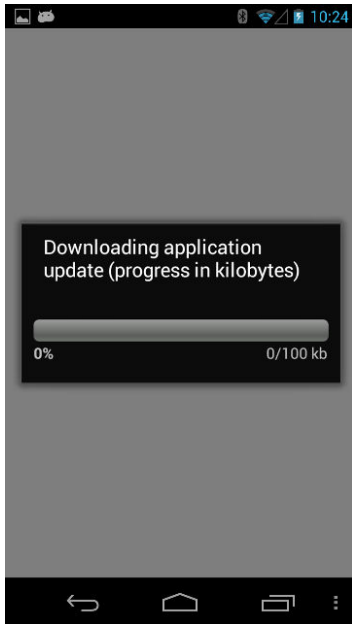
Figure 75. Downloading newly deployed resources to Android
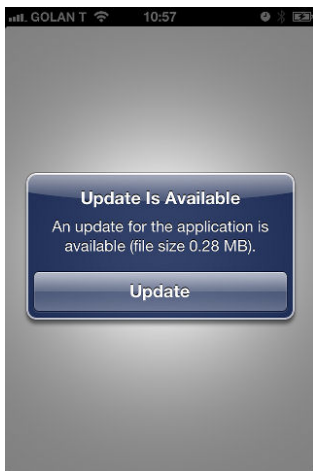
Figure 76. Update notice from iOS
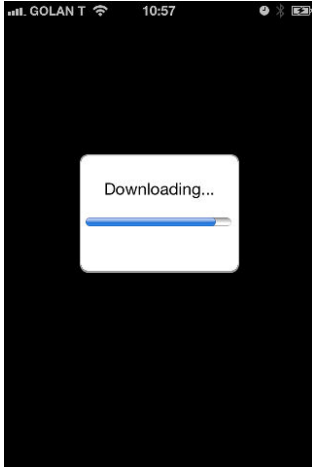
Figure 77. Downloading newly deployed resources to iOS
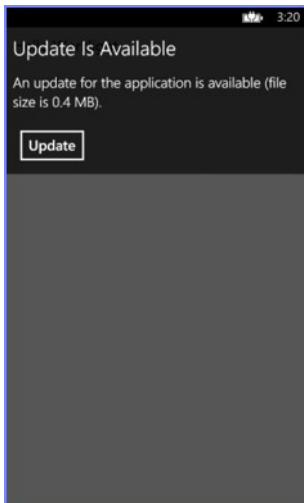
Figure 78. Update notice from Windows Phone 8
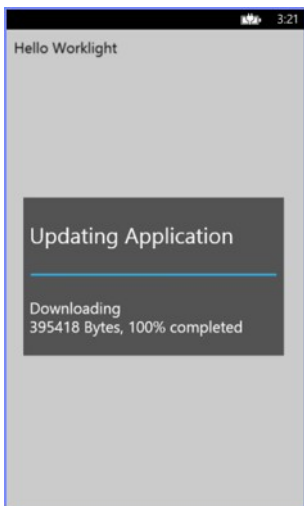
Figure 79. Downloading newly deployed resources to Windows Phone 8

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.

Direct Update as a security realm

Since IBM Worklight Foundation V6.2.0, Direct Update is part of the Worklight security framework, and is defined as a security realm.

The incorporation of Direct Update into the Worklight security framework provides greater flexibility and consistency. Direct Update is enabled by default on all supported platforms, including iPhone, iPad, Android, and Microsoft Windows Phone 8. Checks for Direct Update occur during requests to the server. If an update is available, the client displays a confirmation dialog. When the user accepts, the new resources are downloaded from the Worklight Server and the app restarts.

Configuring the Direct Update realm

You configure Direct Update in the `authenticationConfig.xml` file. A Direct Update test can be added to any mobile security test or custom security test.

Example of a Direct Update test in a mobile security test
```
<mobileSecurityTest name="mobileWithDirectUpdate">
  <testDirectUpdate mode="perRequest"/>
  <testDeviceId provisioningType="none"/>
  <testUser realm="wl_anonymousUserRealm"/>
</mobileSecurityTest>
```

Example of a Direct Update test in a custom security test
```
<customSecurityTest name="customWithDirectUpdate">
  <test realm="wl_directUpdateRealm" mode="perRequest" step="1"/>
  <test realm="wl_anonymousUserRealm" isInternalUserID="true" step="1"/>
  <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>
</customSecurityTest>
```

Mode

You can use the optional `mode` property to configure when the server checks for direct updates. The following table describes the possible modes:

Table 97. Values of the `mode` attribute

Mode	Description
`perSession`	The server checks whether a Direct Update is available (and if so, delivers it) once per session, on the first request to the server. This is the default mode.

Table 97. Values of the `mode` *attribute (continued)*

Mode	Description
`perRequest`	The server checks whether a Direct Update is available (and if so, delivers it) on every request to the server.
`disabled`	The server never checks whether a Direct Update is available (not even during an explicit call to `WL.Client.login`, as described in "Check for Direct Update on demand" on page 605).

Mobile security test

If a Direct Update test is not specified in a mobile security test, it is enabled with the default `perSession` mode. To change the direct update mode to `perRequest` in a mobile security test, add a direct update test with `mode="perRequest"` to a mobile security test: `<testDirectUpdate mode="perRequest"/>`. To disable direct update in a mobile security test, add a direct update test with `mode="disabled"` to the mobile security test: `<testDirectUpdate mode="disabled"/>`.

Custom security test

To add a Direct Update test to a custom security test, add the following test to the security test: `<test realm="wl_directUpdateRealm"/>`. The default mode is `perSession`. To change the mode, specify a value for the `mode` attribute: `<test realm="wl_directUpdateRealm" mode="perRequest"/>`. To disable automatic Direct Update in a custom security test, either set the mode to `disabled` or do not add a test with a Direct Update realm.

Changes from previous versions of IBM Worklight Foundation

In versions earlier than IBM Worklight Foundation V6.2.0, the server checks for direct updates as part of the `WL.Client.connect()` request. This means that it is possible to circumvent direct updates by creating an application with `connectOnStartup=false` and then calling `WL.Client.invokeProcedure()`. The server does not check for direct updates in this case because there is no `WL.Client.connect()` request to process. Since IBM Worklight Foundation V6.2.0, the server checks for updates outside the `WL.Client.connect()` request. Because Direct Update is now part of the security framework, the server can check for direct updates on every request from the client to the server, on first request, or not at all, depending on the configuration.

In versions earlier than IBM Worklight Foundation V6.2.0, returning the application to the foreground triggered a server request to check for direct updates. Since IBM Worklight Foundation V6.2.0, you can configure Worklight Server to check for direct updates on every request; therefore direct updates are no longer explicitly checked when the application returns to the foreground. You can impose the behavior of earlier versions by listening to the "resume" event fired by Cordova and manually checking for direct updates:

```
$(document).on("resume", function(){
    WL.Client.login("wl_directUpdateRealm", {onSuccess:..., onFailure:...});
});
```

Check for Direct Update on demand

You can configure Direct Update so that the server checks for direct updates only when there is an explicit call to do so. To apply this setting, protect the application with a custom security test that does **not** contain a test with `wl_directUpdateRealm`. In the application code, use `WL.Client.login("wl_directUpdateRealm",` `{onSuccess:.., onFailure:...})`, which causes the server to check for direct updates. (See the documentation for the `login` method of `WL.Client`.) This configuration does not work if the custom security test that protects the application contains `wl_directUpdateRealm` with `mode="disabled"`.

Native applications and Direct Update

Direct Update is supported only by the JavaScript WL.Client. For applications that are protected by a direct update realm that is configured to use `perSession` mode, when such applications initiate a server session by using a native `WLClient` call, the server assumes that Direct Update is not required for this session even if subsequent requests are made by using a JavaScript WL.Client call. In such cases, to enable Direct Update to work when using the JavaScript WL.Client, configure the direct update realm to use `perRequest` mode.

Direct Update Customization

Developers can customize the Direct Update process for hybrid applications on iOS, Android, and Microsoft Windows Phone 8. For more information, see "Customizing the direct update interface and process" on page 608.

Upgrading projects

When you import a project from an older version of Worklight Studio into Worklight Studio V6.2.0, the following Direct Update test is added automatically to every custom security test in the project `authenticationConfig.xml` file: `<test realm="wl_directUpdateRealm" step="1"/>`. Direct Update then continues to work in a similar way to how it worked in versions earlier than V6.2 (that is, Direct Update checks are made once per session).

Serving direct update requests from a CDN
You can configure direct update requests to be served from a CDN (content delivery network) instead of from the Worklight Server.

Advantages of using a CDN

Using a CDN instead of the Worklight Server to serve direct update requests has the following advantages:
* Removes network overheads from the Worklight Server.
* Increases transfer rates above the 250 MB/second limit when serving requests from a Worklight Server.
* Ensures a more uniform direct update experience for all users regardless of their geographical location.

General requirements

To serve direct update requests from a CDN, ensure that your configuration conforms to the following conditions:

- The CDN must be a reverse proxy in front of the Worklight Server (or in front of another reverse proxy if needed).
- When building the application from Worklight Studio or from the command-line interface, you need to supply the CDN host and port instead of the host and port of the Worklight Server.
- In the CDN administration panel, you need to mark the following direct update URLs for caching to ensure that the CDN passes all requests to the Worklight Server except for the direct update requests. For direct update requests, the CDN determines if it has already obtained the content. If it has, it returns it without going to the Worklight Server; if not, it goes to the Worklight Server, gets the direct update zip file, and stores it for the next requests for that specific URL.
  - For applications that are built with versions of Worklight Studio earlier than V6.2.0, the direct update URL is: `PROTOCOL://DOMAIN:PORT/CONTEXT_PATH/apps/services/api/APP_NAME/ENVIRONMENT/updates?skin=SKIN_NAME&x-wl-app-version=VERSION`.
  - For applications that are built with Worklight Studio V6.2.0 or later, the direct update URL is: `PROTOCOL://DOMAIN:PORT/CONTEXT_PATH/directUpdate/APP_NAME/ENVIRONMENT/VERSION?skin=SKIN_NAME`.

  Note: The prefix `PROTOCOL://DOMAIN:PORT/CONTEXT_PATH` is constant for all runtime requests and cannot be changed just for direct update requests.
- The CDN must support TTL on the direct update response. This is needed to support multiple direct updates for the same version.
- The CDN must not change or remove the HTTP headers that are used in the Worklight server-client protocol.
- The CDN must allow caching of the request parameters. Two different direct update zip files might differ only by the request parameters (x-wl-app-version or skin).

Example configuration

This example is based on using an Akamai CDN configuration that caches the direct update zip file. The following tasks are completed by the network administrator, the Worklight administrator, and the Akamai administrator:

Network administrator:
1. Create an additional domain in the DNS for your Worklight Server. For example if your server domain is yourcompany.com you need to create an additional domain such as cdn.yourcompany.com.
2. In the DNS for the new cdn.yourcompany.com domain, set a CNAME to the domain name that is provided by Akamai; for example, yourcompany.com.akamai.net.

Worklight administrator:
- Set the new cdn.yourcompany.com domain as the Worklight Server URL for the Worklight applications. For example, for the Ant builder task, the property is: `<property name="wl.server" value="http://cdn.yourcompany.com/${contextPath}/"/>`

Akamai administrator :
1. Open the Akamai property manager and set the property hostname to the value of the new domain.

2. On the Default Rule tab, configure the original Worklight Server host and port, and set the **Custom Forward Host Header** value to the newly created domain.

3. From the **Caching Option** list, select No Store.

4. From the Static Content configuration tab, configure the matching criteria according to the direct update URL of the application. For example, create a condition that states If Path matches one of *direct_update_URL*.

5. Set values similar to the following to configure the caching behavior to make cache the direct update URL and to set TTL.

Table 98. Configuring caching

Field	Value
Caching Option	Cache
Force Revaluation of Stale Objects	Serve stale if unable to validate
Max-age	3 minutes
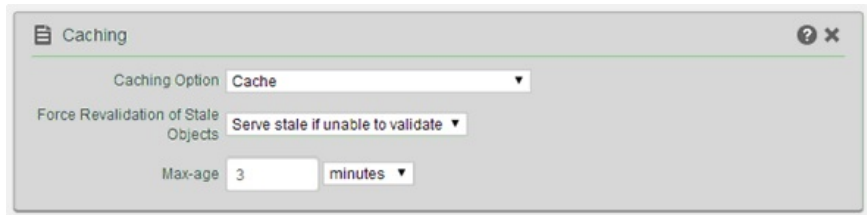
6. Configure the cache key behavior to use all request parameters in the cache key (you must do this to cache different direct update zip files for different applications or versions). For example, from the **Behavior** list, select `Include all parameters (preserve order from request)`.

7. Save and activate the configuration.

Customizing the direct update interface and process

You can change the default user interface for the direct update dialog boxes and the messages that are displayed to the user. You can also control when an application checks for a direct update, run the direct update process without presenting a user interface to the user, link the checking for available direct updates to a call to an adapter, and control what happens when the direct update process fails.

Use the `handleDirectUpdate` function of the direct update challenge handler to customize the direct update process and interface on iOS, Android, and Windows Phone 8. The `handleDirectUpdate` function has the following format:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function (directUpdateData,directUpdateContext)
```

The function accepts the following arguments:

directUpdateData
> A JSON object that contains the `downloadSize` property that represents the files size in bytes of the update package to be downloaded from server.

directUpdateContext
> A JavaScript object that exposes `.start()` and `.stop()` functions that start and stop the direct update flow.

If the web resources are newer on the Worklight Server than in the application, direct update challenge data is added to the server response. Whenever the Worklight client-side framework detects this direct update challenge, it starts the `wl_directUpdateChallengeHandler.handleDirectUpdate` function.

The function provides a default direct update look and feel: a default message dialog that is displayed when a direct update is available and a default progress screen that is displayed when the direct update process is initiated. For examples of default screens, see "Direct updates of app versions to mobile devices" on page 599. You can implement custom direct update user interface behavior or customize the direct update dialog box by overriding this function and implementing your own logic.

The following example `handleDirectUpdate` function implements a custom message in the direct update dialog:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData,directUpdateContext

  var customDialogTitle = 'Custom Title Text';
  var customDialogMessage = 'Custom Message Text';
  var customButtonText = 'Custom Button Text';

  WL.SimpleDialog.show(customDialogTitle, customDialogMessage, [{
    text : customButtonText,
    handler : function() {
      directUpdateContext.start();
    }
  }]);
};
```

You can start the direct update process by running the
directUpdateContext.start() method whenever the user clicks the pop-up dialog
button. This method supports the following types of invocation:

- When no parameters are specified, IBM Worklight Foundation uses the default
  progress screen, which resembles the one in IBM Worklight V6.1.0.
- When a listener function such as
  directUpdateContext.start(directUpdateCustomListener) is supplied, the direct
  update process runs in the background while the process sends lifecycle events
  to the listener. The custom listener must implement the following methods:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){

  },
  onProgress: function(status,totalSize,completedSize){

  },
  onFinish: function(status){

  }
};
```

The listener methods are started during the direct update process according to
following rules:

- onStart is called with the totalSize parameter that holds the size of the update
  file.
- onProgress is called multiple times with status DOWNLOAD_IN_PROGRESS,
  totalSize, and completedSize (the volume that is downloaded so far).
- onProgress is called with status UNZIP_IN_PROGRESS.
- onFinish is called with one of the following final status codes:

Table 99. Status codes for the onFinish rule

Status code	Description
SUCCESS	Direct update finished with no errors.
CANCELED	Direct update was canceled (for example, because the stop() method was called).
FAILURE_NETWORK_PROBLEM	There was a problem with a network connection during the update.
FAILURE_DOWNLOADING	The file was not downloaded completely.
FAILURE_NOT_ENOUGH_SPACE	There is not enough space on the device to download and unpack the update file.
FAILURE_UNZIPPING	There was a problem unpacking the update file.

Table 99. Status codes for the `onFinish` rule (continued)

Status code	Description
FAILURE_ALREADY_IN_PROGRESS	The start method was called while direct update was already running.
FAILURE_UNKNOWN	Unexpected internal error.

If you implement a custom direct update listener, you must ensure that the app is reloaded when the direct update process is complete and the `onFinish()` method has been called. You must also call `wl_directUpdateChalengeHandler.submitFailure()` if the direct update process fails to complete successfully.

The following example shows an implementation of a custom direct update listener:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
  //show custom progress dialog
  },
  onProgress: function(status,totalSize,completedSize){
    //update custom progress dialog
  },
  onFinish: function(status){

    if (status == 'SUCCESS'){
      //show success message
      WL.Client.reloadApp();
    }
    else{
      //show custom error message

      //submitFailure must be called is case of error
      wl_directUpdateChallengeHandler.submitFailure();
    }
  }
};

wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext)

  WL.SimpleDialog.show('Update Avalible', 'Press update button to download version 2.0', [{
    text : 'update',
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Scenario: Running UI-less direct updates

IBM Worklight Foundation supports UI-less direct update when the application is in the foreground.

To run UI-less direct updates, implement `directUpdateCustomListener`. Provide empty function implementations to the `onStart` and `onProgress` methods. Empty implementations cause the direct update process to run in the background.

To complete the direct update process, the application must be reloaded. The following options are available:

- The `onFinish` method can be empty as well. In this case, direct update will apply after the application has restarted.

- You can implement a custom dialog that informs or requires the user to restart the application. (See the following example.)
- The onFinish method can enforce a reload of the application by calling WL.Client.reloadApp().

Here is an example implementation of directUpdateCustomListener:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
  },
  onProgress: function(status,totalSize,completeSize){
  },
  onFinish: function(status){
    WL.SimpleDialog.show('New Update Available', 'Press reload button to update to new version', [
      text : WL.ClientMessages.reload,
      handler : WL.Client.reloadApp
    }]);
  }
};
```

Implement the wl_directUpdateChallengeHandler.handleDirectUpdate function. Pass the directUpdateCustomListener implementation that you have created as a parameter to the function. Make sure directUpdateContext.start(directUpdateCustomListener) is called. Here is an example wl_directUpdateChallengeHandler.handleDirectUpdate implementation:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContex

  directUpdateContext.start(directUpdateCustomListener);
};
```

Note: When the application is sent to the background, the direct-update process is suspended.

Scenario: Triggering direct updates on demand

By default, the application checks for direct updates once per session. You can program the application to check for direct updates at a different point in time, for example, you can trigger a check for direct updates whenever a user clicks a button.

The mobile security test that is provided by default (in the authenticationConfig.xml file under the server/conf folder) contains a direct update security test. You must disable this test if you want to trigger direct update on demand. For example:

Custom security test:
```
<customSecurityTest name="customNoDirectUpdate">
  <test realm="wl_anonymousUserRealm" isInternalUserID="true" step="1"/>
  <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>
</customSecurityTest>
```

Mobile security test:
```
<mobileSecurityTest name="mobileTests">
  <testDeviceId provisioningType="none" />
  <testUser realm="wl_anonymousUserRealm" />
</mobileSecurityTest>
```

In your JavaScript code, when you decided to run direct update (for example, through a WiFi connection or when the application is in the background) call WL.Client.checkForDirectUpdate(). This call triggers direct update on demand.

Scenario: Checking for direct updates when a specific adapter is called

This scenario shows how you can link the checking for direct updates to adapter calls.

Program your application to use a custom security test without direct update. The following example shows such a custom security test in the `authenticationConfig.xml` file:

```xml
<customSecurityTest name="customNoDirectUpdate">
  <test realm="wl_anonymousUserRealm" isInternalUserID="true" step="1"/>
  <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>
</customSecurityTest>
```

Program your adapter to use a custom security test with direct update defined. The following example shows such a custom security test in the `authenticationConfig.xml` file:

```xml
<customSecurityTest name="customWithDirectUpdateRequest">
  <test realm="wl_directUpdateRealm" mode="perRequest"/>
  <test realm="wl_anonymousUserRealm" isInternalUserID="true" step="1"/>
  <test realm="wl_deviceNoProvisioningRealm" isInternalDeviceID="true" step="2"/>
</customSecurityTest>
```

In this case, the application does not require direct update even if it is available on the server until the adapter is called from your JavaScript code as shown in the following example:

```
WL.Client.invokeProcedure({adapter : 'RSSReader', procedure : 'getFeeds'});
```

Scenario: Handling a direct update failure

This scenario shows how to handle a direct update failure that might be caused, for example, by loss of connectivity. In this scenario, the user is prevented from using the app even in offline mode. A dialog is displayed offering the user the option to try again.

Create a global variable to store the direct update context so that you can use it subsequently when the direct update process fails. For example:

```
var savedDirectUpdateContext;
```

Implement a direct update challenge handler. Save the direct update context here. For example:

```
wl_directUpdateChallengeHandler.handleDirectUpdate = function(directUpdateData, directUpdateContext)

  savedDirectUpdateContext = directUpdateContext; // save direct update context

  var downloadSizeInMB = (directUpdateData.downloadSize / 1048576).toFixed(1).replace(".", WL.App.ge
  var directUpdateMsg = WL.Utils.formatString(WL.ClientMessages.directUpdateNotificationMessage, dow

  WL.SimpleDialog.show(WL.ClientMessages.directUpdateNotificationTitle, directUpdateMsg, [{
    text : WL.ClientMessages.update,
    handler : function() {
      directUpdateContext.start(directUpdateCustomListener);
    }
  }]);
};
```

Create a function that starts the direct update process by using the direct update context. For example:

```
restartDirectUpdate = function () {
  savedDirectUpdateContext.start(directUpdateCustomListener); // use saved direct update context t
};
```

Implement `directUpdateCustomListener`. Add status checking in the `onFinish`
method. If the status starts with "FAILURE", open a modal only dialog with the
option "Try Again". For example:

```
var directUpdateCustomListener = {
  onStart: function(totalSize){
    alert('onStart: totalSize = ' + totalSize + 'Byte');
  },
  onProgress: function(status,totalSize,completeSize){
    alert('onProgress: status = ' + status + ' completeSize = ' + completeSize + 'Byte');
  },
  onFinish: function(status){
    alert('onFinish: status = ' + status);
    var pos = status.indexOf("FAILURE");
    if (pos > -1) {
      WL.SimpleDialog.show('Update Failed', 'Press try again button', [ {
        text : "Try Again",
        handler : restartDirectUpdate // restart direct update
      }]);
    }
  }
};
```

When the user clicks the **Try Again** button, the application restarts the direct
update process.

Troubleshooting unintended direct update requests on Windows Phone 8

When the hostname or IP address of the Worklight Server that is configured in a
hybrid Windows Phone 8 application differs from what has been deployed on the
server (either the Worklight Development Server embedded within Worklight
Studio or a production server), the application might receive an unintended direct
update request. If the user proceeds with the direct update, the application is no
longer able to connect to the Worklight Server.

Depending on whether you deploy the application to the Worklight Development
Server or a production server, take one of the following measures to resolve the
problem:

On a production Worklight Server:
> When the IP address of the Worklight Server is modified in an application
> by using the **Build settings and Deploy Target** command in Worklight
> Studio, the resulting `.wlapp` file must be redeployed on the Worklight
> Server.

On the Worklight Development Server:
> The **Build settings and Deploy Target** command is intended to be used for
> modifying an application to connect to a different server. However, it can
> be used to change the IP address of the server and connect to the
> Worklight Development Server that is embedded in Worklight Studio. In
> such cases, the host name of the server must be modified to the
> appropriate IP address and host name. To change it, go to the Servers view
> in Worklight Studio, double-click the Worklight Development Server,
> change the hostname field, and save the settings.

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 Worklight native API.

Similar to other types of mobile applications with IBM Worklight Foundation, 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 Worklight native API:

- The Worklight Objective-C client-side API, if your Native API application is for the iOS environment
- The Worklight Java client-side API, if your Native API application is for the Android environment
- The Worklight Java client-side API, if your Native API application is for the Java Platform, Micro Edition (Java ME)
- The Worklight C# client-side API, if your Native API application is for Windows Phone 8 environment

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**, **Java ME**, or **WindowsPhone8**.
  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**, **Java ME**, or **WindowsPhone8**.
  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 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 Worklight native library.
    - The `worklight.plist` file is the client property file.
  - for Android:
    - The `worklight-android.jar` file defines the 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 Worklight native library.
    - The `wlclient.properties` file is the client property file.
  - for Windows Phone 8:
    - The `worklight-windowsphone8.dll` file defines the Worklight native library. The `Newtonsoft.Json.WindowsPhone.dll` is a library that is required for using JSON objects in C#.
    - 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
- A project in Visual Studio Express for Windows Phone or Visual Studio 2012 Professional or higher, for Windows Phone 8 environment

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 Worklight native API.

- For iOS:
  1. To update the application descriptor file, see "Application Descriptor of Native API applications for iOS" on page 616.
  2. To update the client property file, see "Client property file for iOS" on page 618.
  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 618.
- For Android:
  1. To update the application descriptor file, see "Application Descriptor of Native API application for Android" on page 621.
  2. To update the client property file, see "Client property file for Android" on page 623.

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 624.

- 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 628.
  2. To update the client property file, see "Client property file for Java Platform, Micro Edition (Java ME)" on page 630.
  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 630.

- For Windows Phone 8:
  1. To update the application descriptor file, see "Application Descriptor of Native C# API application for Windows Phone 8" on page 631.
  2. To update the client property file, see "Client property file for Windows Phone 8" on page 632.
  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 Windows Phone 8" on page 633.

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 1195.

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 Foundation 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 Foundation checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM Worklight Foundation 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 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 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.

Chapter 9. Developing Worklight applications **617**

Client property file for iOS
This file defines the properties that your native app for iOS requires to use the 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 you use 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 100. 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 Worklight environment. The value of this property must be `iOSnative`. You must not modify the value of this property value.	`iOSnative`
`languagePreferences`	This property defines a list of preferred languages separated by comma, to be used by Worklight to display system messages. This property is optional.	`en, fr, de, es`
`platformVersion`	This property defines the version number of the Worklight platform.	`6.2.0.00.20140613-0730`
`wlUid`	This property is for internal usage. You must not modify the value.	`wY/mbnwKTDDYQUvuQCdSgg==`

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 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:
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `sqlcipher.framework`
   - `libstdc++.6.dylib`
   - `libz.dylib`

   Note: The framework `sqlcipher.framework` might already be linked.

   Important: If you are using Xcode 7, link `libz.tbd` and `libstdc++.6.tbd`, instead of the corresponding `.dylib` files. Xcode 7 requires the latest interim fix.

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)/WorklightAPI/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.

9. Optional: Set the build options.

Important: If you are using Xcode 7, in the **Build Settings** tab:

a. Open the **Build Options** section.

b. Set **Enable Bitcode** to No.

For more information, see "Disabling bitcode in Xcode builds" on page 621.

Enforcing TLS-secure connections in iOS apps

For development purposes, hybrid iOS projects that are created in Worklight Studio or by using CLI bypass the iOS 9 requirement to enforce Transport Layer Security (TLS) protocol version 1.2 in all apps.

About this task

Apple's App Transport Security (ATS) is a new feature of iOS 9 that enforces best practices for connections between the app and the server. By default, this feature enforces some connection requirements that improve security. These include client-side HTTPS requests and server-side certificates and connection ciphers that conform to Transport Layer Security (TLS) version 1.2 using forward secrecy.

For **development purposes**, you can override the default behavior by specifying an exception in the `Info.plist` file in your app, as described in App Transport Security Technote. However, in a full **production environment**, all iOS apps must enforce TLS-secure connections for them to work properly.

Using iOS 9 and Xcode 7 require the latest interim fix of IBM Worklight Foundation V6.2.0, the apps that you develop in IBM Worklight Foundation V6.0.0. This fix automatically turns off transport security to allow all non-secure connections to the IBM Worklight Foundation Development Server for hybrid apps.

To enable non-TLS connections for native apps, the following exception must appear in the *<projectname>*info.plist file in the *<project>*\Resources folder:

```
<key>NSExceptionDomains</key>
  <dict>
    <key>yourserver.com</key>
    <dict>
      <!--Include to allow subdomains-->
      <key>NSIncludesSubdomains</key>
      <true/>

  <!--Include to allow insecure HTTP requests-->
      <key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
      <true/>
    </dict>
  </dict>
```

Note: If you are creating a **hybrid** iOS application, your Worklight Server host name is added automatically to the `NSException` dictionary as detailed above.

Procedure

1. To prepare for production, remove or comment out the code that appears earlier in this page.

2. Set up the client to send HTTPS requests by using the following entry to the dictionary:

```
<key>protocol</key>
<string>https</string>

<key>port</key>
<string>10443</string>
```

The SSL port number is defined on the server in `server.xml` in the
`httpEndpoint` definition.

3. Configure a server that is enabled for the TLS 1.2 protocol. For more
   information, see Configuring MobileFirst Server to enable TLS V1.2.
4. Make settings for ciphers and certificates, as they apply to your setup. For
   more information, see App Transport Security Technote, Secure communications
   using Secure Sockets Layer (SSL) for WebSphere Application Server Network
   Deployment, and Enabling SSL communication for the Liberty profile.

Disabling bitcode in Xcode builds

You must disable the new bitcode option in Xcode builds for IBM Worklight
Foundation projects.

About this task

Starting with Xcode 7, bitcode is a default, but optional option for iOS apps. The
bitcode option is not currently supported in IBM Worklight Foundation. To use the
Worklight SDK in your Xcode projects, you must disable bitcode.

Note: Applications that are based on Apple watchOS 2 require the bitcode to be
enabled and are currently not supported in IBM Worklight Foundation. Using
Xcode 7 and iOS 9 requires the latest Worklight interim fix.

Procedure

On the **Xcode Build Settings** tab, in the **Build Options** group, set **Enable Bitcode**
to **No**.

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 Foundation 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 Foundation checks whether the client is
        already authenticated according to the security test. If the client is not yet
        authenticated, IBM Worklight Foundation 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
        theWorklight 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 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 about how to extract this file, see "Extracting a public signing key from native apps" on page 625.

```
</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 Worklight native API for Android.

The `wlclient.properties` client property file contains the necessary data to use the 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 101. 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 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.	
`languagePreferences`	This property defines a list of preferred languages separated by comma, to be used by Worklight to display system messages.	`en, fr, de, es`

Property	Description	Example values
`wlUid`	This property is for internal usage. You must not modify the value.	`wY/mbnwKTDDYQUvuQCdSgg==`

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 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, the `bcprov.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 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. This permission is required if you are targeting your apps for Android API level 13 and below.
   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. Optional: If JSONStore is required, copy the contents of the `jsonstore/assets/` and `jsonstore/libs/` folders into your application's `assets/` and `libs/` folders, respectively.

7. 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, you must create a `splash.png` file or `splash.9.png` file and place it 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.

   Note: If you create a hybrid Android app, the `splash.9.png` file is automatically created. If you do not want to use the splash screen in your app, you must delete it from these `drawable` folders.

8. Optional: If app authenticity is required, copy the `armeabi\libauthjni.so`, `armeabi-v7a\libauthjni.so`, `mips\libauthjni.so`, and `x86\libauthjni.so` folders into your application's `libs/` folder.

Extracting a public signing key from native apps

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

Procedure

1. In the Eclipse project explorer, right-click your Native API folder created for the Android environment and click the **Extract public signing key** menu item.

A wizard window opens.
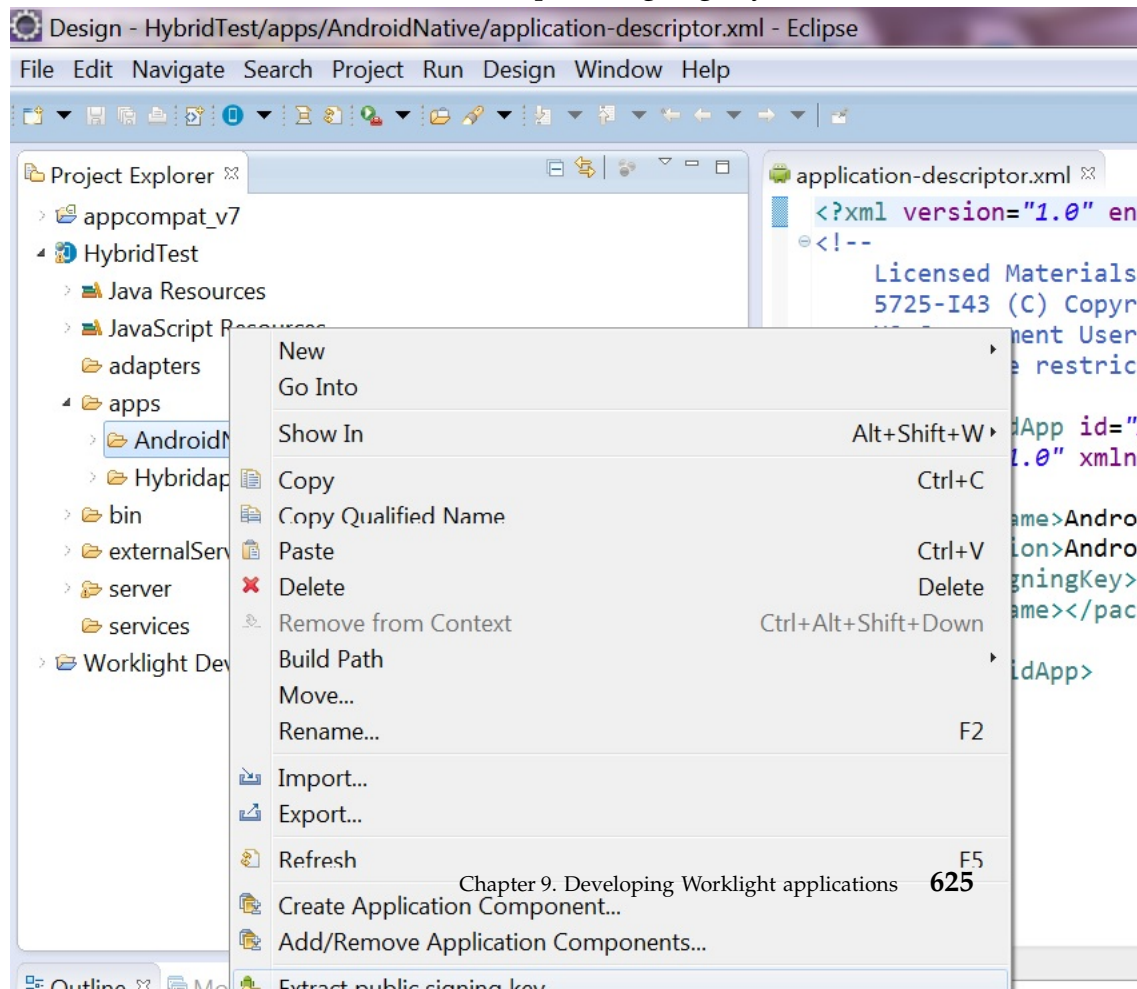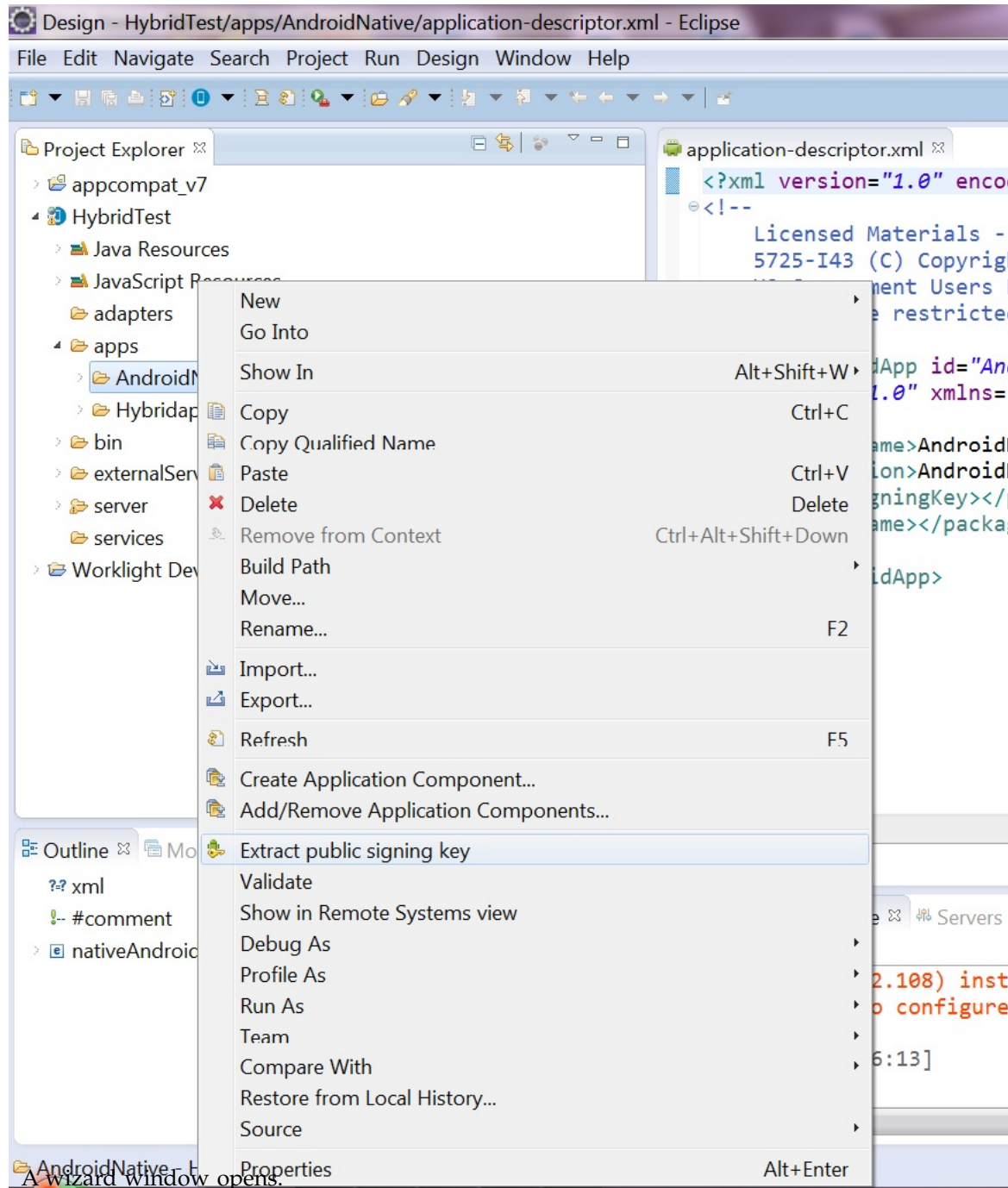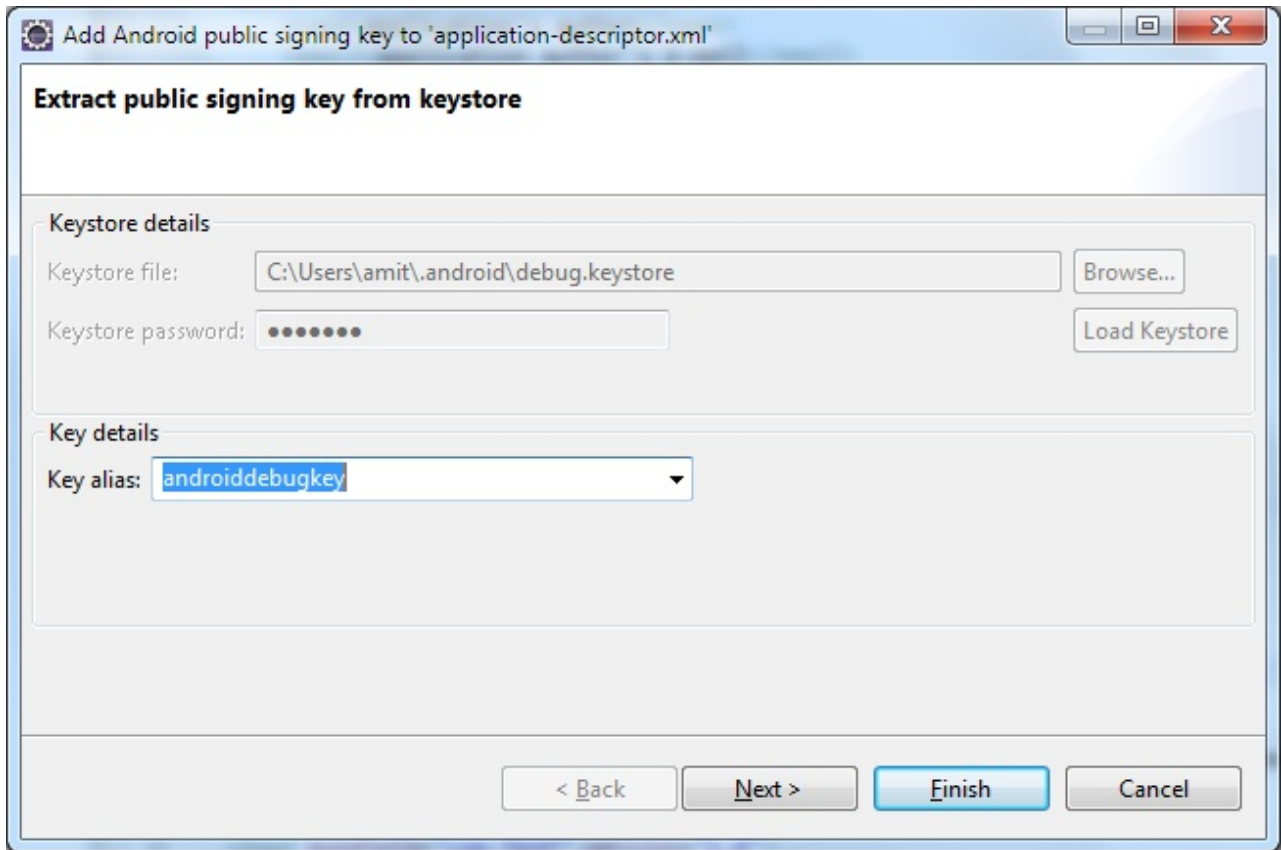
Figure 80. Adding the Android public signing key

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	`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.

Figure 81. 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="false"/>
  <security>
    <testAppAuthenticity enabled="false"/>
    <encryptWebResources enabled="false"/>
    <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
    <publicSigningKey>MIGfMA0CSqGSIb3DQEBAQUAA4GNADCBiQKBgQCE+TiHbDxPx0HA6rARXoJWC071hLLBytTDSdNe/></publicSigningKey>
  </security>
</android>
```

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 Foundation 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 Foundation checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM Worklight Foundation 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 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 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 Worklight native API for Java ME.

The `wlclient.properties` client property file contains the necessary data to use the 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 102. 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 Worklight environment. The value of this property must be `JavaMEnative`. You must not modify the value of this property value.	`JavaMEnative`
`languagePreferences`	This property defines a list of preferred languages separated by comma, to be used by Worklight to display system messages.	`en, fr, de, es`

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 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.

Developing native C# applications for Windows Phone 8

This collection of topics gives instructions for developing native C# applications for Windows Phone 8.

Application Descriptor of Native C# API application for Windows Phone 8

The application descriptor is a metadata file that is used to define various aspects of the native C# API application for Windows Phone 8.

The application descriptor is in the application root directory, and its name is application-descriptor.xml. You must define the various aspects of the application in this metadata file.

The following example shows the format of the application descriptor file of Native C# API applications for Windows Phone 8:

```
<?xml version="1.0" encoding="UTF-8"?>
<nativeWindowsPhone8App
  xmlns="http://www.worklight.com/native-windowsphone8-descriptor">
  id="WP8"
  platformVersion="6.2.0"
  version="1.0"
  <displayName>application display name</displayName>
  <description>application description</description>
  <pushSender>
    <authenticatedPush serviceName="MPNS Service name" keyAlias="certificate alias" keyAliasPasswo
  </pushSender>

</nativeWindowsPhone8App>
```

The <nativeWindowsPhone8App> element is the root element of the descriptor. This element has three mandatory 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 C# or JavaScript.

platformVersion
	This attribute contains the version of IBM Worklight Foundation 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.

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

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

`<pushSender>`

`<pushSender>`
> This element indicates to the Worklight Server that the app is designed to receive push notifications. It also contains information necessary to provide to MPNS for sending authenticated push notifications.

`<authenticatedPush serviceName="MPNS Service name" keyAlias="certificate alias" keyAliasPassword="`

`<authenticatedPush>`

> This element has three attributes:

> **serviceName**
>> The common name (CN) found in the subject value of the MPNS certificate.

> **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.

`</nativeWindowsPhone8App>`

`</nativeWindowsPhone8App>`
> This tag closes the content of the application descriptor file.

Client property file for Windows Phone 8

This file defines the properties that your native C# app for Windows Phone 8 requires to use the Worklight native C# API for Windows Phone 8.

The `wlclient.properties` client property file contains the necessary data to use the Worklight API for Windows Phone 8.

You must define the properties of this client property file before you use it in your native C# app for Windows Phone 8.

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

Table 103. 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`

Property	Description	Example values
`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 Worklight environment. The value of this property must be `WindowsPhone8native`. You must not modify the value of this property.	`WindowsPhone8native`
`wlMPNSServiceName`	The service name to be used for authenticated push as specified in the application descriptor. For more information, see "Application Descriptor of Native C# API application for Windows Phone 8" on page 631.	`MyServiceName`
`languagePreferences`	This property defines a list of preferred languages separated by comma, to be used by Worklight to display system messages.	`en, fr, de, es`

Copying files of Native API applications for Windows Phone 8

To copy the files in the Native C# API application for Windows Phone 8 into the project that defines the native C# app for Windows Phone 8.

About this task

To use the Worklight Native C# API for Windows Phone 8 in your native C# app, you must copy the library and the client property file of your Native C# API application into your native C# app for Windows Phone 8 project.

Procedure

In your Microsoft Visual Studio project for the native C# app for Windows Phone 8:

1. Add the `worklight-windowsphone8.dll` file as a reference to your Visual Studio project.
2. Add the `Newtonsoft.Json.WindowsPhone.dll` file as a reference to your Visual Studio Project.
3. Copy the `wlclient.properties` client property file from the Native C# API application, and paste it into your Visual Studio project.
4. Right-click the `wlclient.properties` file and select **Properties**.

5. In the **Properties** window, set the **Copy to Output Directory** option to **Copy always**.

6. Add the following capabilities to the `WMAppManifest.xml` file of your Windows Phone 8 Visual Studio project:

   a. `ID_CAP_NETWORKING`

   b. `ID_CAP_IDENTITY_DEVICE`

   c. `ID_CAP_PUSH_NOTIFICATION`, if the native app is used for push notifications.

Adding Worklight web capabilities to an existing native app

You can add a Worklight web view into an existing native Android, iOS, or Windows Phone 8 application. For example, you can transform the application from pure native to hybrid. With such a change, you can add Worklight web capabilities to an existing native application.

About this task

This is a two-step process.

1. Export the corresponding resources from an existing Worklight project.

2. Integrate those resources into your native app and update your code to show a Worklight web view.

Procedure

1. Using either the Eclipse plug-in or the command-line interface, export the corresponding resources from an existing Worklight hybrid project. Note that when first adding the web view to a native app, you must select the option to export the native libraries. Subsequent exports do not need to include these libraries and will result in much smaller archives.

   - **By using the Worklight Studio Eclipse plug-in:**

     a. Ensure that the latest resources are built.

     b. Right-click **Run As: Build Environment**.

     c. Right-click the environment folder. For example, right-click **iphone**, **android**, or **windowsphone8**.

     d. Select **Export Worklight Hybrid Resources**.

     e. Specify the location in which to create the `.zip` file archive. For example, `hybrid_resources.zip`.

   - **By using the Worklight Studio command-line interface**:

     a. Change the directory to the environment folder. For example, `cd apps/myhybridapp/android`.

     b. Make sure that the latest resources are built, for example, wl build.

     c. Run the **wl export** command.

     d. Specify the name and location in which to create the `.zip` archive. For example, `hybrid_resources.zip`.

2. Integrate the hybrid resources into the native application to show a Worklight web view. Here are the steps for Android, iOS, and Windows Phone 8 devices.

   - **For Android devices**

     a. For Android devices, extract the archive, for example `hybrid_resources.zip` file, to the native Android project.

     b. If you are using the Android Native API, some files might need to be replaced, such as the libraries (`lib/*.jar`) and the client properties (`assets/wlclient.properties`).

c. When you are extracting the file, if your application contains a file with the same path name as provided by the Worklight Hybrid archive, merge the contents of the two files such as `res/` and `values/strings.xml`.

d. Update the `AndroidManifest.xml` file for your native Android application with the contents of the `AndroidManifest-WL.xml` file, which is provided in the Worklight hybrid archive.

e. If you are using Android Studio, take into account that the project directory might differ from the Eclipse ADT. Extract the Worklight hybrid archive content to the `app/libs` directory for libraries and the `app/src/main/` directory for assets and resources.

f. Update your code to initialize the Worklight Web Framework and show a Cordova web view. See the sample in "Implementing a custom startup process in Android-based hybrid applications" on page 461.

- **For iOS devices**

  a. If the files `WorklightAPI` and `worklight.plist` are in your iOS native project, delete them.

  b. Extract the Worklight hybrid resources contents from the archive, for example the `hybrid_resources.zip` file, into the directory that contains your iOS native project where the `.xcodeproj` file is located.

  c. Add the Cordova subproject to the native application:

     1) Right-click **Project Navigator - Add Files to <Xcde_project_name>**

     2) Select **CordovaLib/CordovaLib.xcodeproj**.

     3) Select **Copy items into destination group's folder (if needed)**.

     4) Select **Create groups for any added folders**.

  d. Add the Worklight files to the native application:

     1) Right-click **Project Navigator - Add Files to <Xcde_project_name>**.

     2) Select the following files and directories:

        ```
        buildtime.sh
        config.xml
        FipsHttp
        Frameworks/
        Resources/
        Tealeaf/
        worklight.plist
        WorklightSDK
        ```

     3) Select **Copy items into destination group's folder (if needed)**.

     4) Select **Create groups for any added folders**.

  e. Add the web assets to the native application:

     1) Right-click **Project Navigator - Add Files to <Xcode_project_name>**.

     2) Select `www/`.

     3) Select **Copy items into destination group's folder (if needed)**.

     4) Select **Create groups for any added folders**.

  f. Add the Cordova library as a build dependency:

     1) Select **Target**.

     2) Select **Build Phases**.

     3) Under **Target Dependencies**, add **CordovaLib**.

  g. Add the Cordova library dependency, which is `libCordova.a`.

  h. Add **Run Script Before Copy Bundle Resources**.

  i. Add **Run Script as last Phase**.

  j. Select **Editor** > **Add Build Phase** > **Add Run Script Build Phase**.

k. Open a shell and, at the command prompt, type:

```
/bin/sh
touch -cm ${PROJECT_DIR}/www
```

l. Add **Run Script After Copy Bundle Resources**.

m. Select **Editor** > **Add Build Phase** > **Add Run Script Build Phase**.

n. Open a shell and, at the command prompt, type:

```
/bin/sh
script_file="buildtime.sh"
echo "Running a custom build phase script: $script_file"
unsecure_project_path=${PROJECT_DIR}
secure_project_path="${unsecure_project_path// /\ }"
eval ${secure_project_path}/${script_file}
scriptExitStatus=$?
echo "DONE with script: ${script_file} (exitStatus=${scriptExitStatus})\n\n"
exit "${scriptExitStatus}"
```

o. Add Linker flag to link the external dependencies:

1) Select **Target**.

2) Click **Build Settings**.

3) Expand **Linking** and select **All**.

4) Double-click the value for **Other Linker Flags**.

   Tip: Use the search box to filter the list.

5) Copy and paste the following code:

```
-force_load
"$(BUILT_PRODUCTS_DIR)/libCordova.a"
-force_load
"$(SRCROOT)/WorklightSDK/libWorklightStaticLibProject.a"
-ObjC
```

p. Check the armv7 architecture.

1) Select **Project**.

2) Click **Build Settings**.

3) Expand the Architectures section.

4) Click a value for **Architectures**.

5) Select **Other**.

6) Remove **Standard architectures (including 64-bit)** or
   $ARCHS_STANDARD_INCLUDING_64_BIT.

7) Add **armv7**.

q. Verify that the following libraries and frameworks are referenced:

```
AddressBook.framework
AddressBookUI.framework
AssetsLibrary.framework
AudioToolbox.framework
AVFoundation.framework
CFNetwork.framework
CoreData.framework
CoreGraphics.framework
CoreLocation.framework
CoreMedia.framework
CoreMotion.framework
CoreTelephony.framework
Foundation.framework
ImageIO.framework
MediaPlayer.framework
MobileCoreServices.framework
Security.framework
SystemConfiguration.framework
```

```
QuartzCore.framework
UIKit.framework
libz.dylib
libstdc++.6.dylib
```

> **Important:** If you are using Xcode 7, link `libz.tbd` and `libstdc++.6.tbd`, instead of the corresponding `.dylib` files. Xcode 7 requires the latest interim fix.

r. Update your code to initialize the Worklight Web Framework and show a Cordova web view. See the sample in "Implementing a custom startup process for iOS-based Hybrid applications" on page 455.

- **For Windows Phone 8 devices**

  a. Extract the hybrid resources, for example the `hybrid_resources.zip` file, into a directory, such as `WP8HybridResources`.

  b. Add the hybrid resources to the native application by completing the following steps:

    1) Open the native application in Visual Studio.

    2) Add the `www` directory to the native application.

    3) Add the `config.xml` file to the native application.

    4) Add the `worklight.xml` file to the native application properties.

    5) Add the `wlclient.properties` file to the native application.

    6) Add the `WLWPNativeLib.dll` and `WLCordovaClassLib.dll` files to the native application.

    7) If the native application does not already contain the `Newtonsoft.Json.WindowsPhone.dll` file, then add the `NewtonSoft.Json.dll` file to the native application.

    8) Right-click **References** and select **Add References**. Add the `WLWPNativeLib.dll` and `WLCordovaClassLib.dll` files as copied. If the Native Application does not contain the `Newtonsoft.Json.WindowsPhone.dll` reference, then add the reference to the copied `NewtonSoft.Json.dll` file.

  c. Update the capabilities from the hybrid application:

    1) From the native application in Visual Studio, open **Native App** > **Properties** > **WMAppManifest.xml**.

    2) Add the missing capabilities, which are part of the `WP8HybridResources/Properties/WMAppManifest.xml` file.

  d. Create a page and copy the contents from the hybrid application:

    1) Add a new page from native project in Visual Studio.

    2) Right-click the native project and select **Add** > **New Item** > **Windows phone portrait page**.

    3) Provide a name for the page such as `HybridPage.xaml`.

  e. Open the new page, which you might have called `HybridPage.xaml`, and remove the exiting Grid XML content.

  f. Open the `WP8HybridResources/MainPage.xaml` page in a text editor.

  g. Copy the Grid XML section and paste it into the new page, which you might have called `HybridPage.xaml`. The following code shows the new grid XML section:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
  <Grid.RowDefinitions>
    <RowDefinition Height="*"/>
  </Grid.RowDefinitions>
  <my:CordovaViewHorizontalAlignment="Stretch"
```

```
                Margin="0,0,0,0"
                x:Name="CordovaView"
                VerticalAlignment="Stretch"
                StartPageUri="/www/skinLoader.html"/>
        </Grid>
```

h. Copy the `xmlns:mytag` from the `WP8HybridResources/Mainpage.xaml` file to the `HybridPage.xaml` page for the phone tag. For example:

```
xmlns:my="clr-namespace:WPCordovaClassLib;
assembly=WPCordovaClassLib"
```

i. Open the `App.xaml.cs` file of the native application in Visual Studio.

j. In the `App Constructor` method, after the `lineInitializePhoneApplicaton` method, add the following code to initialize the native library:

```
new WPNativeLib.Initializer();
```

k. Open the `HybridPage.xaml.cs` file of the native application in Visual Studio.

l. After the constructor, add the following method:

```
voidCordovaView_Loaded(object sender, RoutedEventArgs e)
{
}
```

m. Open the hybrid page from the native application.

n. Add a button in the native application.

o. For the `Click` method, write the following code to open the hybrid page:

```
NavigationService.Navigate(new Uri("/HybridPage.xaml", UriKind.Relative));
```

p. **Optional**: If you use the JSONSTORE feature from the JavaScript API on Windows Phone 8, perform the following steps:

1) Copy the `WP8HybridResources/buildtarget` directory into the native application root folder.

2) Open the `<native_application>`.csproj file, which is the Visual Studio project file for the native application.

3) Copy the following XML code into this project file:

```
<ItemGroup>
  <Content Include="buildtarget\$(Platform)\sqlite3.dll">
  <Link>sqlite3.dll</Link>
  <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </Content>
  <Reference Include="JSONStoreWP8Lib">
    <HintPath>buildtarget\$(Platform)\JSONStoreWP8Lib.dll</HintPath>
  </Reference>
</ItemGroup>
```

4) Open the native application in Visual Studio. An application with JSONStore enabled supports x86 and ARM architectures. To specify a specific platform, complete the following steps:

   – Right-click the solution node within the Solution Explorer.

   – Select **Properties**.

   – Select the **Configuration Properties** within the tree to the left in the **Properties Page** window.

   – Click **Configuration Manager**, which opens a new window.

   – Select your targeted architecture, such as **x86** or **ARM**, but not **Any CPU**.

5) In Visual Studio, open the `App.xaml.cs` file of the native application.

6) In the `App Constructor` method, after the new `WPNativeLib.Initializer()` line, add the following code to initialize the JSONStore library:

Updating mobile apps with IBM Worklight Foundation and the Application Center

You can choose among different ways to update a mobile application depending on the context.

When you build applications for internal use in your organization and that are not to be delivered through public app stores such as Google play or Apple App Store, you could use the Application Center to deliver these applications over the air. There are several scenarios that you might want to consider.

Changing the web part of a hybrid application

When you want to deliver a Worklight hybrid application that consists mainly of HTML5 with CSS and JavaScript, and you must change the hybrid part of the application to provide new features or to fix a defect, you do not have to ask the application users to update it on their devices. You can use the Worklight direct update mechanism to deploy new HTML with CSS and JavaScript for your application without changing the application version on the mobile device. To perform this kind of update, you do not recreate a native version of the application. You only redeploy the Worklight application (wlapp) in the Worklight Console for the same application version. See "Deploying applications and adapters to Worklight Server" on page 1195.

You can implement this type of update in a way that the application user has nothing to do to trigger it. Alternatively, you can provide a menu entry in your application to check for available updates and to trigger the direct update mechanism. For more details, see "Direct Update as a security realm" on page 603.

If you use the direct update mechanism, you will not have to redeploy a new binary version of your application to the Application Center. When new users install applications from the Application Center, they get the latest version of the hybrid part of the application through the direct update mechanism when they start the application for the first time or when they use the menu option that is implemented in the application to check for updates.

Delivering a new version of native code

The main reason that you would want to deliver a new version of an application is probably because your application uses native code and you want to provide new features or deliver fixes that require changes in the native code. You might also need to provide a new native version of the application, even if your Worklight application is completely written by using web technologies, to accommodate new mobile operating systems supported only in later versions of IBM Worklight. You cannot use the direct update mechanism in either of these cases. You must build and deploy a new version of the application.

You can provide the update through the Application Center.

Start by uploading the new application binary (APK, IPA, or other) to the Application Center console. The application is listed as a new available version of the application. For the Application Center to consider the application as a new version of an existing application, you must keep the application identification unchanged; for example, it must have the same package name for Android or the

same bundle ID for iOS. You change the internal version of the application; for example, **versionCode** on Android or **CFBundleVersion** on iOS. For more information about application properties in the Application Center console, see "Application properties" on page 1322.

If you configured the Application Center to send push notifications for updates, users would receive a notification as soon as the new version is deployed. The content of this message enables the user to open the Application Center mobile client on the update page of the mobile application, so that he or she can trigger an over-the-air installation.

To streamline the update process, you can remotely disable the previous version of the Worklight application from the Worklight Console. See "Remotely disabling application connectivity" on page 1249. By remotely disabling the previous version, you can make sure that your users do not use the old version anymore.

When you disable the previous version, you must provide the URL of the latest application version, so that users have an easy way to fetch the new version.

In the Application Center, if you want to direct users of the mobile application to the update page of the mobile client, you can use a custom URL of the format:

`ibmappctr://show-app? Package-name`

Where *Package-name* is the package name of the application that you have updated to a new version.

The exact URL is listed in the "Application properties" page of the Application Center console. For more information, see "Application properties" on page 1322.

When a user launches a disabled application version, the user is directed to get the update on the main page of the application in the Application Center mobile client. An Update button gives access to over-the-air update of the application.

Updating the Application Center application

Since IBM Worklight Foundation V6.2.0, when a new version of the Application Center becomes available, users do not have to uninstall the mobile application before downloading and installing the new version on their mobile devices. For example, when a new version of the Application Center is made available to support new mobile operating systems. Users can be automatically notified when a new version of the mobile client is available in the Application Center repository. In your role as Administrator of the Application Center, you have only to upload the new binary version of the mobile client application to the catalog.

Command Line Interface for IBM Worklight Developers

IBM Worklight Foundation provides command-line tools as an alternative to the integrated development environment (IDE) of Worklight Studio.

To help developers get a better tools experience, IBM Worklight Foundation provides a command-line interface (CLI) tool to easily create and manage both native and hybrid apps. The CLI enables developers to use their preferred text editors or alternative IDEs to create mobile applications.

The command-line interface does not require Worklight Studio for most standard activities. The commands support tasks such as creating, adding, and configuring

with the Worklight API library, adding the client-side Worklight properties file and performing the build and deploy of the Worklight application. Adapter creation, deployment, and local testing can be performed within the command line. Administration of your Worklight project can be done from CLI or REST services, or the Console, where you can easily control the local server and observe the logs. Command-line tools can be used on their own, or in parallel with theWorklight Studio tools.

Everything that is generated by using the command-line interface is compatible with Worklight Studio. You can also use the CLI to integrate third-party tools such as ANT or Grunt to create your own tool chain for automated testing, build, and deployment flows.

To install command-line tools, see "Installing command-line tools for developers" on page 64.

Overview of Worklight CLI commands

IBM Worklight Foundation provides the available subcommands and their options for the Command Line Interface for IBM Worklight Developers.

Overview of available sub commands

Using your CLI to run the commands, you can use either **wl** or **worklight**.

NAME

IBM Worklight Command-Line Interface (CLI).

SYNOPSIS

`wl command [options]`

DESCRIPTION

Command-line interface to create and manage the apps for IBM Worklight Foundation.

Global Commands

`wl create [<name>]`

This command creates a new Worklight project in the current working directory.

`wl create-server`

This command creates a new Liberty server in your default folder.
The server is pre-configured to work as a Worklight local test server.

`wl console`

This command opens the Worklight Console in your default browser for the current project.

wl help [<command>]

This command displays all of the Syntax summary or command help.

Project-Level Commands

add [adapter|api|hybrid|feature|skin options]

This command generates various Worklight artifacts.

wl add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]

This command creates a new adapter that is generated into the adapters folder of the current

wl add api [<name> --environment|-e <environment>]

This command creates a new Native API that is generated into the apps folder of the current

wl add environment [--type|-t <type>[,...]] [--app|-a <app>]]

This command adds a platform-specific environment to a hybrid application.
You can run wl add environment (without any arguments). Prompts then request the environm

wl add feature [fips|jsonstore|tealeaf]

This command adds optional features to your hybrid applications. When used, it enters an in

wl add hybrid [<name>]

This command creates a new hybrid application, which is generated into the apps folder of the

wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]

This command is a way of creating Worklight skins within your hybrid applications.

wl build

This command builds the set of Worklight resources that are most local to the current working

wl deploy

This command deploys to the local test server the set of Worklight resources that are most loc

wl export [<path to zip file> [-i|--includeNativeLibs]]

This command permits you to create a zip file, which contains either the entire Worklight proj
Note: This command requires that the current working directory is under an existing hybrid ap

wl invoke [<adapter>:<procedure> [\"<arguments>\"]]

This command invokes a procedure for a specified adapter on the Worklight Server.

wl remove feature [fips|jsonstore|tealeaf]

This command removes optional features from your hybrid applications. When used, it ente

wl run

This command starts the local test server. It is a long running process, and displays new sta
The run command does not return control to the command line.

wl start

This command starts the local test server. On completion, a message indicates a successful s

wl status

This command shows the status (running or stopped) of the local test server.

wl stop

This command stops the local test server.

Command-line Flags/Options

-v, --version   Prints this utility's version
-d, --debug   Debug mode produces verbose log output

Example usage

$ wl create WLProject
$ cd WLProject
$ wl add api MyiOS --platform ios
$ wl add adapter MySQLAdapter --type sql
$ cd MySQLAdapter
$ wl build
$ wl deploy

CLI commands and their definitions

The CLI commands are intended for IT developers for creating Worklight projects
separately from the Eclipse Worklight Studio.

Here is a compact list of the CLI commands with descriptions of their function, see
"Commands." All Worklight CLI commands can be run interactively, or directly if
all required options are provided on the command line.

Commands
Use the command-line interface (CLI) keywords and options to work with IBM
Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:
- The direct method: You enter the command and set its options on one line and
  press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press
  Enter. For example, **$ wl add adapter**. Then, you are prompted to set the
  available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]
> The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]
> The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.
>
> If you run **wl add adapter** without any arguments, you are prompted for the following parameters:
>
> **<name>**
>> The name that you want for the generated adapter.
>
> **<adaptertype>**
>> The type of adapter. For example: HTTP, Cast Iron.
>
> **[--jsonstore|-j]**
>> Your choice of JSONStore procedures.
>
> **[--ussd|-u]**
>> The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]
> The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.
>
> If you run **wl add api** without any arguments, you are prompted for the following parameters:
>
> **<name>**
>> The name that you want for the generated native API.
>
> **-e [ios|android|javame|windowsphone8]**
>> The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]
> This command adds a platform-specific environment to a hybrid application.
>
> If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:
>
> - To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
> - To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
> The **add feature** command is a way of adding optional features to your

hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
FIPS 140-2

jsonstore
JSONStore

tealeaf
IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
The **add skin** command is a way of creating Worklight skins in your hybrid applications.

Note: Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
You can run the command either of the following ways:

- Interactive: **$ wl add skin**. You are prompted for the following parameters:
  - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
  - Skin name: Enter a name for the skin.
- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.

- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
Use the **config** command to set your configuration preferences, in

particular which browser is used by Command Line Interface for IBM
Worklight Developers. To configure your preferences, you can use one of
the following syntaxes:

- Interactive: Run **$ wl config** and follow the prompts by using the arrow
  keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>]
  [<value>]**.

CONSOLE $ wl console

The **console** command opens the Worklight Console in your default
browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by
running **wl create-server** and that this server is running when you use
the **console** command.

This command takes no parameters. To open the Worklight Console run: **$
wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current
working directory. Run the command in the current working directory. If
you enter the command without any arguments, you are prompted for the
name of the project. To generate a new Worklight project called MyProject,
enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The **create server** command creates a new WebSphere Application Server
Liberty application server in your default folder. The server is
preconfigured to work as a Worklight local test server. To run the
command, enter **wl create-server**. This command takes no additional
arguments.

D:

DEPLOY $ wl deploy

The **deploy** command deploys to the local test server the set of Worklight
resources that are most local to the current working directory. To run the
deploy command in the preferred folder, navigate to the folder and enter **$
wl deploy**.

- If you run the command from the root folder of a Worklight project, the
  command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all
  the apps in the project.
- If you run the command from the folder of a specific adapter, the
  command deploys that one adapter.
- If you run the command from the folder of a specific application, the
  command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the
entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a string parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  > FIPS 140-2

  jsonstore
  > JSONStore

  tealeaf
  > IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

The **stop** command stops the local test server.

Note: Prerequisite: Make sure that the local test server exists.
Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:

Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

* The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
* The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.

If you run **wl add adapter** without any arguments, you are prompted for the following parameters:

<name>
The name that you want for the generated adapter.

<adaptertype>
The type of adapter. For example: HTTP, Cast Iron.

[--jsonstore|-j]
Your choice of JSONStore procedures.

[--ussd|-u]
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]
The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.

If you run **wl add api** without any arguments, you are prompted for the following parameters:

<name>
The name that you want for the generated native API.

-e [ios|android|javame|windowsphone8]
The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a
<app>][ipad|iphone|android|windowsphone8][,...]
This command adds a platform-specific environment to a hybrid application.

If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:

- To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
- To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the **[?]** prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
FIPS 140-2

jsonstore
JSONStore

> **tealeaf**
>> IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
>> The **add skin** command is a way of creating Worklight skins in your hybrid applications.
>>
>> **Note:** Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
>> You can run the command either of the following ways:
>> - Interactive: **$ wl add skin**. You are prompted for the following parameters:
>>   - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
>>   - Skin name: Enter a name for the skin.
>> - Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
>> The **build** command builds the set of Worklight resources that are most local to the current working directory.
>>
>> **Note:** Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.
>>
>> To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
>> - If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
>> - If you run the command from the apps folder, the command builds all the apps in the project.
>> - If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
>> Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:
>> - Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
>> - Direct with the **setting** parameter: **$ wl config [<setting>]**.
>> - Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console
>> The **console** command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy

The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you

enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  FIPS 140-2

  jsonstore
  JSONStore

  tealeaf
  IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to

the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

The **stop** command stops the local test server.

Note: Prerequisite: Make sure that the local test server exists.
Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:

Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.

If you run **wl add adapter** without any arguments, you are prompted for the following parameters:

<name>
The name that you want for the generated adapter.

<adaptertype>
The type of adapter. For example: HTTP, Cast Iron.

[--jsonstore|-j]
Your choice of JSONStore procedures.

[--ussd|-u]
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]
The **add api** command generates a new native API into the apps folder of

the current project. Run the command in the current working directory, which is a child of a Worklight project.

If you run **wl add api** without any arguments, you are prompted for the following parameters:

<name>
> The name that you want for the generated native API.

-e [ios|android|javame|windowsphone8]
> The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]
> This command adds a platform-specific environment to a hybrid application.
>
> If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:
> - To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
> - To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
> The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.
>
> The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.
>
> **Note:** The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.
>
> **fips**
> > FIPS 140-2
>
> **jsonstore**
> > JSONStore
>
> **tealeaf**
> > IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
> The **add skin** command is a way of creating Worklight skins in your hybrid applications.
>
> **Note:** Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
> You can run the command either of the following ways:
> - Interactive: **$ wl add skin**. You are prompted for the following parameters:

- Environment type: Select Android smartphones and tablets, iPad, or iPhone.
- Skin name: Enter a name for the skin.
- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build

The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]

Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:
- Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console

The **console** command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is

preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy

The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a string parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  FIPS 140-2

  jsonstore
  JSONStore

  tealeaf
  IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

The **stop** command stops the local test server.

Note: Prerequisite: Make sure that the local test server exists.
Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:
Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: `add [adapter|api|hybrid] [<options>]`
- The interactive method: You enter the command with no arguments and press Enter. For example, `$ wl add adapter`. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]
> The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]
> The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.
>
> If you run **wl add adapter** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated adapter.
>
> **<adaptertype>**
> > The type of adapter. For example: HTTP, Cast Iron.
>
> **[--jsonstore|-j]**
> > Your choice of JSONStore procedures.
>
> **[--ussd|-u]**
> > The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]
> The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.
>
> If you run **wl add api** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated native API.
>
> **-e [ios|android|javame|windowsphone8]**
> > The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]
> This command adds a platform-specific environment to a hybrid application.

If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:

- To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
- To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]

The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
    FIPS 140-2

jsonstore
    JSONStore

tealeaf
    IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]

The **add skin** command is a way of creating Worklight skins in your hybrid applications.

Note: Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
You can run the command either of the following ways:

- Interactive: **$ wl add skin**. You are prompted for the following parameters:
  - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
  - Skin name: Enter a name for the skin.
- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build

The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.

- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:

- Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console
The **console** command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>
The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server
The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy
The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.

- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  FIPS 140-2

jsonstore
> JSONStore

tealeaf
> IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart
> The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run
> The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start
> The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status
> The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop
> The **stop** command stops the local test server.
>
> **Note:** Prerequisite: Make sure that the local test server exists.
> Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:
Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]
> The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]
> The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.
>
> If you run **wl add adapter** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated adapter.
>
> **<adaptertype>**
> > The type of adapter. For example: HTTP, Cast Iron.
>
> **[--jsonstore|-j]**
> > Your choice of JSONStore procedures.
>
> **[--ussd|-u]**
> > The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]
> The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.
>
> If you run **wl add api** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated native API.
>
> **-e [ios|android|javame|windowsphone8]**
> > The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]
> This command adds a platform-specific environment to a hybrid application.
>
> If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:
> - To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
> - To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
> The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
    FIPS 140-2

jsonstore
    JSONStore

tealeaf
    IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
The **add skin** command is a way of creating Worklight skins in your hybrid applications.

Note: Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
You can run the command either of the following ways:

- Interactive: **$ wl add skin**. You are prompted for the following parameters:
    - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
    - Skin name: Enter a name for the skin.
- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:

- Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console

The **console** command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy

The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

> The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:
>
> - Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
> - Direct with no parameters: **$ wl invoke adapterName:function**.
> - Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
> - Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

> The **remove feature** command removes optional features from your hybrid applications.
>
> **Note:** Prerequisite: The current working directory must be under an existing hybrid application.
> To run the command in one the following syntaxes:
>
> - Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.
>
>   **fips**
>   > FIPS 140-2
>
>   **jsonstore**
>   > JSONStore
>
>   **tealeaf**
>   > IBM Tealeaf SDK
>
> - Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

> The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

> The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command

does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

>The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

>The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

>The **stop** command stops the local test server.

>**Note:** Prerequisite: Make sure that the local test server exists.
>Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:

Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

>The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

>The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.

>If you run **wl add adapter** without any arguments, you are prompted for the following parameters:

>>**<name>**
>>>The name that you want for the generated adapter.

\<adaptertype\>
The type of adapter. For example: HTTP, Cast Iron.

[--jsonstore|-j]
Your choice of JSONStore procedures.

[--ussd|-u]
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [\<name\> -e [ipad|iphone|android|javame|windowsphone8]]
The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.

If you run **wl add api** without any arguments, you are prompted for the following parameters:

\<name\>
The name that you want for the generated native API.

-e [ios|android|javame|windowsphone8]
The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a \<app\>][ipad|iphone|android|windowsphone8][,...]
This command adds a platform-specific environment to a hybrid application.

If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:

- To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
- To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
FIPS 140-2

jsonstore
JSONStore

tealeaf
IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
>The **add skin** command is a way of creating Worklight skins in your hybrid applications.
>
>**Note:** Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
>You can run the command either of the following ways:
>
>* Interactive: **$ wl add skin**. You are prompted for the following parameters:
>   - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
>   - Skin name: Enter a name for the skin.
>* Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
>The **build** command builds the set of Worklight resources that are most local to the current working directory.
>
>**Note:** Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.
>
>To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
>
>* If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
>* If you run the command from the apps folder, the command builds all the apps in the project.
>* If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
>Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:
>
>* Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
>* Direct with the **setting** parameter: **$ wl config [<setting>]**.
>* Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console
>The **console** command opens the Worklight Console in your default browser for the current project.
>
>**Note:** Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>
The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called `MyProject`, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server
The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy
The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]
The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]
The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter

and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
      FIPS 140-2

  jsonstore
      JSONStore

  tealeaf
      IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

      The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

      The **stop** command stops the local test server.

      Note: Prerequisite: Make sure that the local test server exists.
      Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:

Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

      The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

      The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.

      If you run **wl add adapter** without any arguments, you are prompted for the following parameters:

      <name>
          The name that you want for the generated adapter.

      <adaptertype>
          The type of adapter. For example: HTTP, Cast Iron.

      [--jsonstore|-j]
          Your choice of JSONStore procedures.

      [--ussd|-u]
          The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]

      The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.

If you run **wl add api** without any arguments, you are prompted for the following parameters:

<name>
> The name that you want for the generated native API.

-e [ios|android|javame|windowsphone8]
> The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]
> This command adds a platform-specific environment to a hybrid application.

> If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:
> - To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
> - To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
> The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

> The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

> **Note:** The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
> FIPS 140-2

jsonstore
> JSONStore

tealeaf
> IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
> The **add skin** command is a way of creating Worklight skins in your hybrid applications.

> **Note:** Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
> You can run the command either of the following ways:
> - Interactive: **$ wl add skin**. You are prompted for the following parameters:
>   - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
>   - Skin name: Enter a name for the skin.

- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build

The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]

Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:
- Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console

The **console** command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy
> The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.
> - If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
> - If you run the command from the apps folder, the command deploys all the apps in the project.
> - If you run the command from the folder of a specific adapter, the command deploys that one adapter.
> - If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]
> The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.
>
> **Note:** Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.
> - If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
> - If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]
> The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:
> - Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
> - Direct with no parameters: **$ wl invoke adapterName:function**.
> - Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
> - Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]
> The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  > FIPS 140-2

  jsonstore
  > JSONStore

  tealeaf
  > IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart
> The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run
> The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start
> The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status
> The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop
> The **stop** command stops the local test server.
>
> **Note:** Prerequisite: Make sure that the local test server exists.
> Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:
Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:

- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

> The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

> The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.
>
> If you run **wl add adapter** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated adapter.
>
> **<adaptertype>**
> > The type of adapter. For example: HTTP, Cast Iron.
>
> **[--jsonstore|-j]**
> > Your choice of JSONStore procedures.
>
> **[--ussd|-u]**
> > The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]

> The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.
>
> If you run **wl add api** without any arguments, you are prompted for the following parameters:
>
> **<name>**
> > The name that you want for the generated native API.
>
> **-e [ios|android|javame|windowsphone8]**
> > The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]

> This command adds a platform-specific environment to a hybrid application.
>
> If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:

- To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
- To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]
The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
FIPS 140-2

jsonstore
JSONStore

tealeaf
IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
The **add skin** command is a way of creating Worklight skins in your hybrid applications.

Note: Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
You can run the command either of the following ways:
- Interactive: **$ wl add skin**. You are prompted for the following parameters:
  - Environment type: Select Android smartphones and tablets, iPad, or iPhone.
  - Skin name: Enter a name for the skin.
- Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
The **build** command builds the set of Worklight resources that are most local to the current working directory.

Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
- If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.

- If you run the command from the apps folder, the command builds all the apps in the project.
- If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]

Use the `config` command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:

- Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
- Direct with the **setting** parameter: **$ wl config [<setting>]**.
- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console

The `console` command opens the Worklight Console in your default browser for the current project.

Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the `console` command.

This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called `MyProject`, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

The `create server` command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy

The `deploy` command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the `deploy` command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.

- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:

- Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ wl invoke adapterName:function**.
- Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
- Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

The **remove feature** command removes optional features from your hybrid applications.

Note: Prerequisite: The current working directory must be under an existing hybrid application.
To run the command in one the following syntaxes:

- Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.

  fips
  > FIPS 140-2

  jsonstore
  > JSONStore

  tealeaf
  > IBM Tealeaf SDK

- Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

> The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

> The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

> The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

> The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

> The **stop** command stops the local test server.
>
> **Note:** Prerequisite: Make sure that the local test server exists.
> Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Commands:
Use the command-line interface (CLI) keywords and options to work with IBM Worklight Foundation from a command prompt window.

You can run the commands in either of the following ways:
- The direct method: You enter the command and set its options on one line and press Enter. For example: **add [adapter|api|hybrid] [<options>]**
- The interactive method: You enter the command with no arguments and press Enter. For example, **$ wl add adapter**. Then, you are prompted to set the available parameters one by one.

A:

ADD add [adapter|api|hybrid] [<options>]

> The **add** command generates new Worklight artifacts. The current working directory must be a child of an existing Worklight project. Generated artifacts go into the appropriate folder within the project, regardless of the

current working directory. For example, adapters are generated to the adapters folder and native APIs are generated to the apps folder.

ADD ADAPTER add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j] [--ussd|-u]]

The **add adapter** command creates a new adapter, which is generated into the adapters folder of the current project.

If you run **wl add adapter** without any arguments, you are prompted for the following parameters:

<name>
The name that you want for the generated adapter.

<adaptertype>
The type of adapter. For example: HTTP, Cast Iron.

[--jsonstore|-j]
Your choice of JSONStore procedures.

[--ussd|-u]
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

ADD API add api [<name> -e [ipad|iphone|android|javame|windowsphone8]]

The **add api** command generates a new native API into the apps folder of the current project. Run the command in the current working directory, which is a child of a Worklight project.

If you run **wl add api** without any arguments, you are prompted for the following parameters:

<name>
The name that you want for the generated native API.

-e [ios|android|javame|windowsphone8]
The environment or mobile platform.

ADD ENVIRONMENT $ wl add environment [--app|-a <app>][ipad|iphone|android|windowsphone8][,...]

This command adds a platform-specific environment to a hybrid application.

If you run **wl add environment** without any arguments, you are prompted for the parameters. You can add multiple environment types, separated by commas. Examples:

- To add an environment when only one hybrid application exists in the Worklight project, or when you work in the directory of a hybrid application, write the command as follows: **$ wl add environment iphone,android**
- To add an environment that specifies the hybrid application, write the command as follows: **$ wl add environment iphone,android --app myHybridApps**

ADD FEATURE $ wl add feature [fips|jsonstore|tealeaf]

The **add feature** command is a way of adding optional features to your hybrid applications. When you use it, it enters an interactive mode where the prompt displays a list of available features: **[fips|jsonstore|tealeaf]**.

The current working directory must be under an existing hybrid application. If you run **$ wl add feature** without any arguments, enter the feature option at the [?] prompt.

Note: The features that were previously installed are not displayed for selection. Use the arrow keys to select a feature and press Enter to add it.

fips
    FIPS 140-2

jsonstore
    JSONStore

tealeaf
    IBM Tealeaf SDK

ADD SKIN $ wl add skin [--environment|-e android|blackberry|blackberry10|iphone|ipad <name>]
    The **add skin** command is a way of creating Worklight skins in your hybrid applications.

    Note: Prerequisite: Make sure that the current working directory is under an existing hybrid application and that at least one environment is added to your hybrid app.
    You can run the command either of the following ways:

    • Interactive: **$ wl add skin**. You are prompted for the following parameters:
        – Environment type: Select Android smartphones and tablets, iPad, or iPhone.
        – Skin name: Enter a name for the skin.
    • Direct: Enter **$ wl add skin [<your_app_environment> <you_skin_name>]**.

B:

BUILD $ wl build
    The **build** command builds the set of Worklight resources that are most local to the current working directory.

    Note: Prerequisite: Make sure that the local test server exists. To create the server, run **wl create-server**.

    To generate a build in the preferred folder, navigate to the folder and enter **$ wl build**.
    • If you run the command from the root folder of a Worklight project, the command builds all the apps and adapters in the project.
    • If you run the command from the apps folder, the command builds all the apps in the project.
    • If you run the command from the folder of a specific adapter, the command builds that one adapter.

C:

CONFIG $ wl config [<setting>][<value>]
    Use the **config** command to set your configuration preferences, in particular which browser is used by Command Line Interface for IBM Worklight Developers. To configure your preferences, you can use one of the following syntaxes:
    • Interactive: Run **$ wl config** and follow the prompts by using the arrow keys.
    • Direct with the **setting** parameter: **$ wl config [<setting>]**.

- Direct with the **setting** and **value** parameters: **$ wl [<setting>] [<value>]**.

CONSOLE $ wl console

    The **console** command opens the Worklight Console in your default browser for the current project.

    Note: Prerequisite: Make sure that the local test server is created by running **wl create-server** and that this server is running when you use the **console** command.

    This command takes no parameters. To open the Worklight Console run: **$ wl console**.

CREATE $ wl create <name>

    The create command creates a new Worklight project in the current working directory. Run the command in the current working directory. If you enter the command without any arguments, you are prompted for the name of the project. To generate a new Worklight project called MyProject, enter **$ wl create MyProject**.

CREATE SERVER $ wl create-server

    The **create server** command creates a new WebSphere Application Server Liberty application server in your default folder. The server is preconfigured to work as a Worklight local test server. To run the command, enter **wl create-server**. This command takes no additional arguments.

D:

DEPLOY $ wl deploy

    The **deploy** command deploys to the local test server the set of Worklight resources that are most local to the current working directory. To run the **deploy** command in the preferred folder, navigate to the folder and enter **$ wl deploy**.

- If you run the command from the root folder of a Worklight project, the command deploys all apps and adapters in the project.
- If you run the command from the apps folder, the command deploys all the apps in the project.
- If you run the command from the folder of a specific adapter, the command deploys that one adapter.
- If you run the command from the folder of a specific application, the command deploys that one application.

E:

EXPORT [<path to zip file>] [-i|--includeNativeLibs]]

    The **export** command creates a compressed file, which can contain the entire Worklight project or the hybrid assets for use in a native application.

    Note: Prerequisite: The current working directory must be under an existing hybrid application or in the root directory. Otherwise, the command returns an error message.

- If your current working directory is in a Worklight project root folder, the source files for the entire project are exported.
- If your current working directory is an environment folder inside a hybrid application, the hybrid web assets are exported for that environment.

I:

INVOKE [<adapter>:<procedure> [\"<arguments>\"|[[--file-f] <path to JSON array file>]]]

> The **invoke** command starts a procedure for a specified adapter on Worklight Server. Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure that are based on what is available in the current working directory of your project. You can run the command with one of the following syntaxes:
>
> - Interactive: **$ wl invoke**, then follow the prompts by using the arrow keys.
> - Direct with no parameters: **$ wl invoke adapterName:function**.
> - Direct with a `string` parameter: **$ wl invoke adapterName:function [\"String\", 2, true]**.
> - Direct with a JSON file parameter: **$ wl invoke adapterName:function --file ./myArts.json**.

R:

REMOVE FEATURE $ wl remove feature [fips|jsonstore|tealeaf]

> The **remove feature** command removes optional features from your hybrid applications.
>
> **Note:** Prerequisite: The current working directory must be under an existing hybrid application.
> To run the command in one the following syntaxes:
>
> - Interactive: **$ wl remove feature**: the [?] prompt shows what features you can remove. Use the arrow keys to select the feature and press Enter to remove it.
>
>   **fips**
>   > FIPS 140-2
>
>   **jsonstore**
>   > JSONStore
>
>   **tealeaf**
>   > IBM Tealeaf SDK
>
> - Direct: **$ wl remove feature [fips|jsonstore|tealeaf]**.

RESTART $ wl restart

> The **restart** command restarts the local test server. Run the **restart** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, and control is returned to the command line. This command takes no arguments. To run the **restart** command in the preferred folder, navigate to the folder and enter **$ wl restart**.

RUN $ wl run

> The **run** command starts the local test server. Run **wl run** in the current working directory, which is a child of a Worklight project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped. The **run** command takes no arguments. To run it, enter**$ wl run**.

S:

START $ wl start

> The **start** command starts the local test server. Run **wl start** command in the current working directory, which is a child of a Worklight project. On completion, a message indicates a successful start, control is returned to the command line, and the Worklight Console is opened in your default browser. This command takes no arguments. To start the local test server, enter **$ wl start**.

STATUS $ wl status

> The **status** command shows the status of the local test server, either running or stopped. Run the command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl status**.

STOP $ wl stop

> The **stop** command stops the local test server.

> **Note:** Prerequisite: Make sure that the local test server exists.
> Run **wl stop** command in the current working directory, which is a child of a Worklight project. This command takes no arguments. To run the command, enter **$ wl stop**.

Accelerating application development by reusing resources

Use application components, 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 519.

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 Foundation searches for Worklight project templates and application components whenever you add an application component to a 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 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 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 104. 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 Worklight project resources that were selected when the application component was created, as well as a mandatory `COMPONENT-DATA` folder.
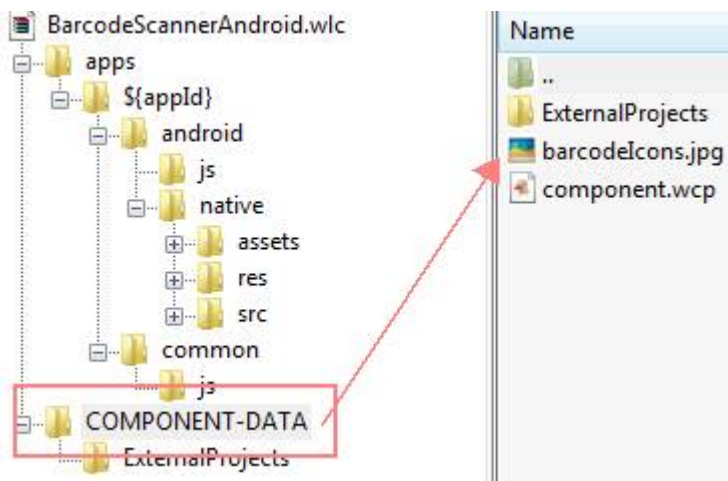
Figure 82. File structure of a typical application component

The `COMPONENT-DATA` folder contains the following files:
- The thumbnail image file that was selected when the application component was created.
- A 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
- 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 a 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 105. 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 106 on page 690 lists the inner elements that are supported on Android and the order in which they must appear in the schema.

Table 106. 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 107 lists the inner elements that are supported on iOS and the order in which they must appear in the schema.

Table 107. 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 a Worklight project, the CordovaPlugin properties are automatically inserted into the `config.xml` file.

Element name

`<CordovaPlugin>`

Parameters

Table 108. 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
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 109. 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 110. 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 a 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 Worklight project that imports the component.

Element name

```
<Receivers>
```

Parameters

Table 111. 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 Worklight project that adds the application component.

Element name

`<Strings>`

Parameters

Table 112. 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 Worklight project that imports the component.

Element name

`<Libraries>`

Parameters

Table 113. 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 83 on page 698 shows an updated android\native\libs folder based on the example <Libraries> element.

Figure 83. 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 a Worklight project. Typically, these libraries are additional libraries that are not packaged as part of the application component.

Element name

```
<ExternalLibraries>
```

Parameters

Table 114. 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 a 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 115. 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 a 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 116. 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 a 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 689 that describe the syntax.

Adding application components to 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 Worklight projects" on page 414.)

2. Set up new 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 Worklight environment for your application" on page 438.)

   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 686.)

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 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.

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 Worklight project templates

You can create Worklight project templates by exporting 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 117. 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 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
- Worklight version used to create the template
- Template ID
- Template author
- Template description
- Template thumbnail
- 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 Worklight projects from 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 a Worklight project from a 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 Worklight projects" on page 414.
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 686.)
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 420 and "Developing native applications" on page 614) 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 Worklight application 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 4, "Tutorials and samples," on page 35.
- 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 4, "Tutorials and samples," on page 35.

- 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 4, "Tutorials and samples," on page 35.

- 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 4, "Tutorials and samples," on page 35.

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, if you add a local instance of Worklight Server, name it "Group Test Server" and designate it as your default test server, this command appears in the menus as **Run As** > **Run on Group 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 12, "Deploying Worklight projects," on page 1101.

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 442.

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, name it "Group Test Server" and designate it as your default test server, this command appears as **Run As** > **Run on Group Test Server**. For consistency, when referring to this menu command in this IBM Worklight Foundation 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**:
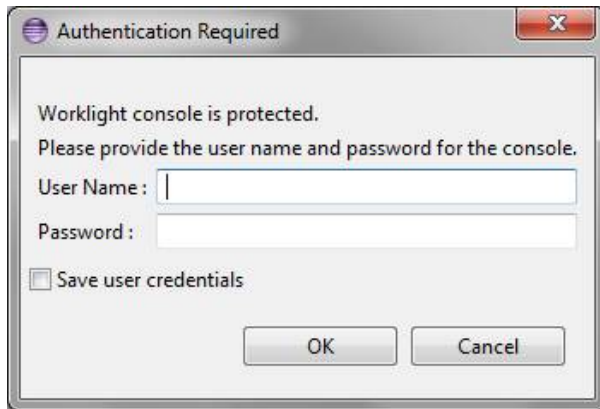
If you enter the correct credentials, the deployment continues when you click **OK**.

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.

OutOfMemoryError exceptions

If you get an OutOfMemoryError exception while deploying a large application to the Worklight Development Server, consider increasing the heap size of the server. To increase the heap size, edit the `jvm.options` file by double-clicking it in the Servers view in Worklight Studio, then either increase the value of the flag `-Xmx` if it already exists, or add this flag with the value 1024m if it does not exist yet: `-Xmx1024m`.

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 Foundation 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.

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 a 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 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:

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, using either the **Run As** > **Run on Worklight Development Server** or the **Run As** > **Build...** menu commands.

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.
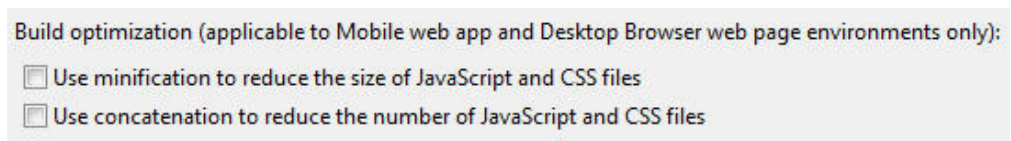
You use this area of the dialog if you want to change the build settings for Desktop Browser and Mobile Web environments for the currently selected server, and want to apply these new minification and concatenation settings to future builds.

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 "Worklight application build settings" on page 723.
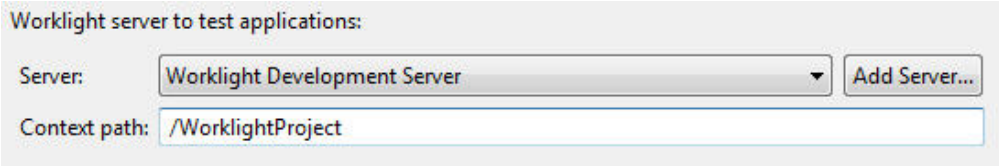
For more information about minification, see "Minification of JS and CSS files" on page 726.

For more information about concatenation, see "Concatenation of JS and CSS files" on page 728.

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.

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, name it "Group Test Server" and designate it as your default test server, this command appears in the menus as **Run As** > **Run on Group 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**:
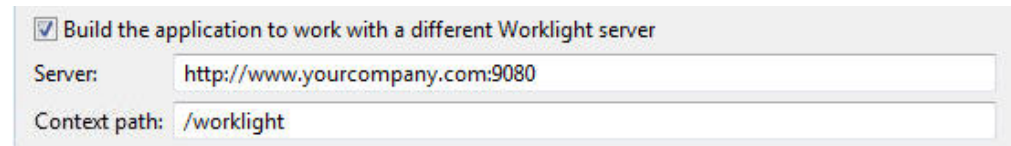
The **Server** field is required, and contains the URL for the remote target server. The entry must use the format: `http(s)://<hostname>:<port>`.

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 1105.

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.
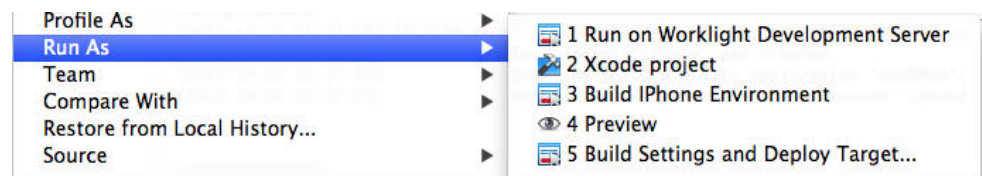
For more information about the iOS development environment, see the module *Previewing your application on iOS*, under category 2, *Hello Worklight*, in Chapter 4, "Tutorials and samples," on page 35.

- 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.
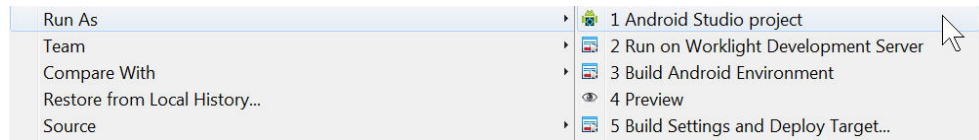
Run As	▸	📱 1 Android Studio project
Team	▸	🖥 2 Run on Worklight Development Server
Compare With	▸	🖥 3 Build Android Environment
Restore from Local History...		👁 4 Preview
Source	▸	🖥 5 Build Settings and Deploy Target...

For more information about the Android development environment, see the module *Previewing your application on Android*, under category 2, *Hello Worklight*, in Chapter 4, "Tutorials and samples," on page 35.

- 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.
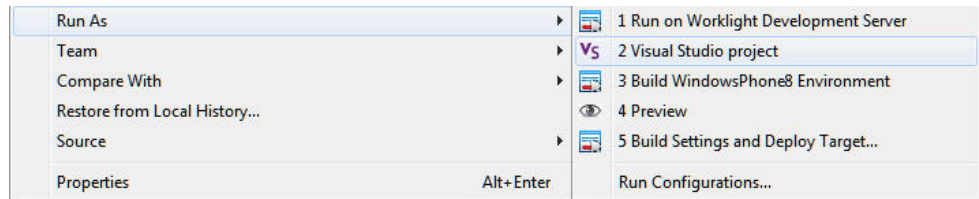
Run As	▸	🖥 1 Run on Worklight Development Server
Team	▸	Vs 2 Visual Studio project
Compare With	▸	🖥 3 Build WindowsPhone8 Environment
Restore from Local History...		👁 4 Preview
Source	▸	🖥 5 Build Settings and Deploy Target...
Properties	Alt+Enter	Run Configurations...

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 4, "Tutorials and samples," on page 35.

Optimizing 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 Foundation, you can include or exclude features from the application build if those features are not required. For example, JSONStore offers many benefits, if code that references it is actually used in the application. If it is not used, the JSONStore resources greatly increase the application size, and thus slow both initial app download time and app start time.

There is a new `<features>` element in the Application Descriptor that controls the inclusion or exclusion of resources. In the `application-descriptor.xml` file itself, this element appears similar to the following example, which shows JSONStore resources being included in the build:

```
<application xmlns="http://www.worklight.com/application-descriptor" id="myApp" platformVersion="6
    ...
    <features>
        <JSONStore/>
    </features>
    ...
</application>
```

For more information about Application Descriptor attributes, see "The application descriptor" on page 430.

When you first create a Worklight application, the `<features>` tag is automatically created in the `application-descriptor.xml` file, with no contents. What this means is that if you use JSONStore in your code, it is not automatically added to the builds. When you run the application, you receive an error, as shown in the following screen capture:
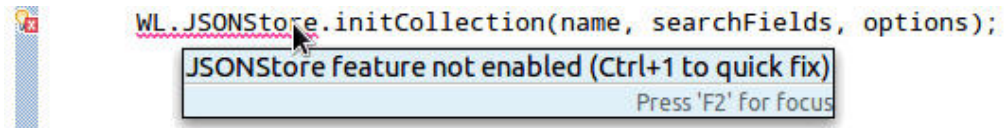
You can resolve this situation by using an Eclipse QuickFix:
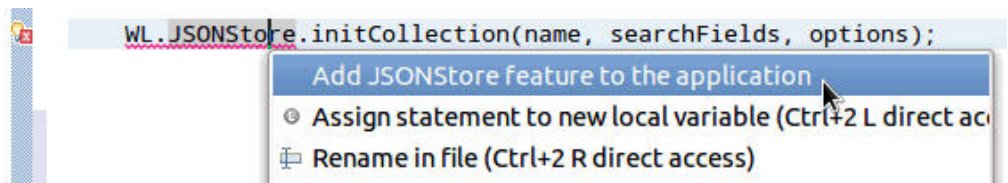
But you can also choose which features to include in the build with the Worklight Studio editor, as shown in the following procedure.

To include or exclude features in Worklight Studio

1. In Worklight Studio, open the `application-descriptor.xml` file for your application with the Application Descriptor Editor:
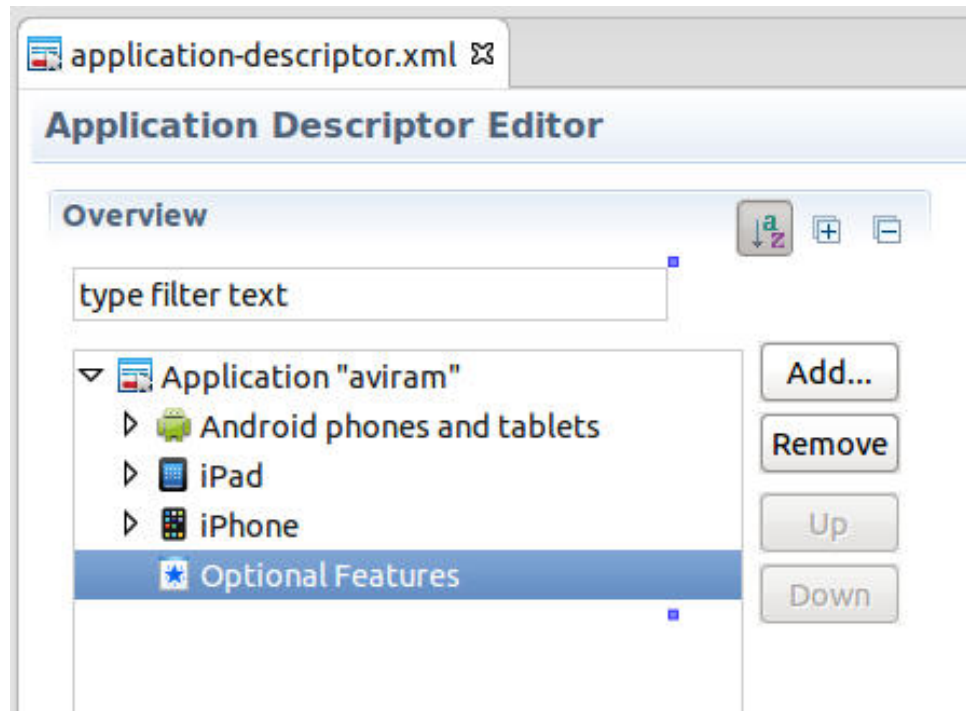
If the **Optional Features** element is empty (as in the screen capture), no
features such as JSONStore are included in the build.

2. To add features, click **Add** to display the Add Item window:
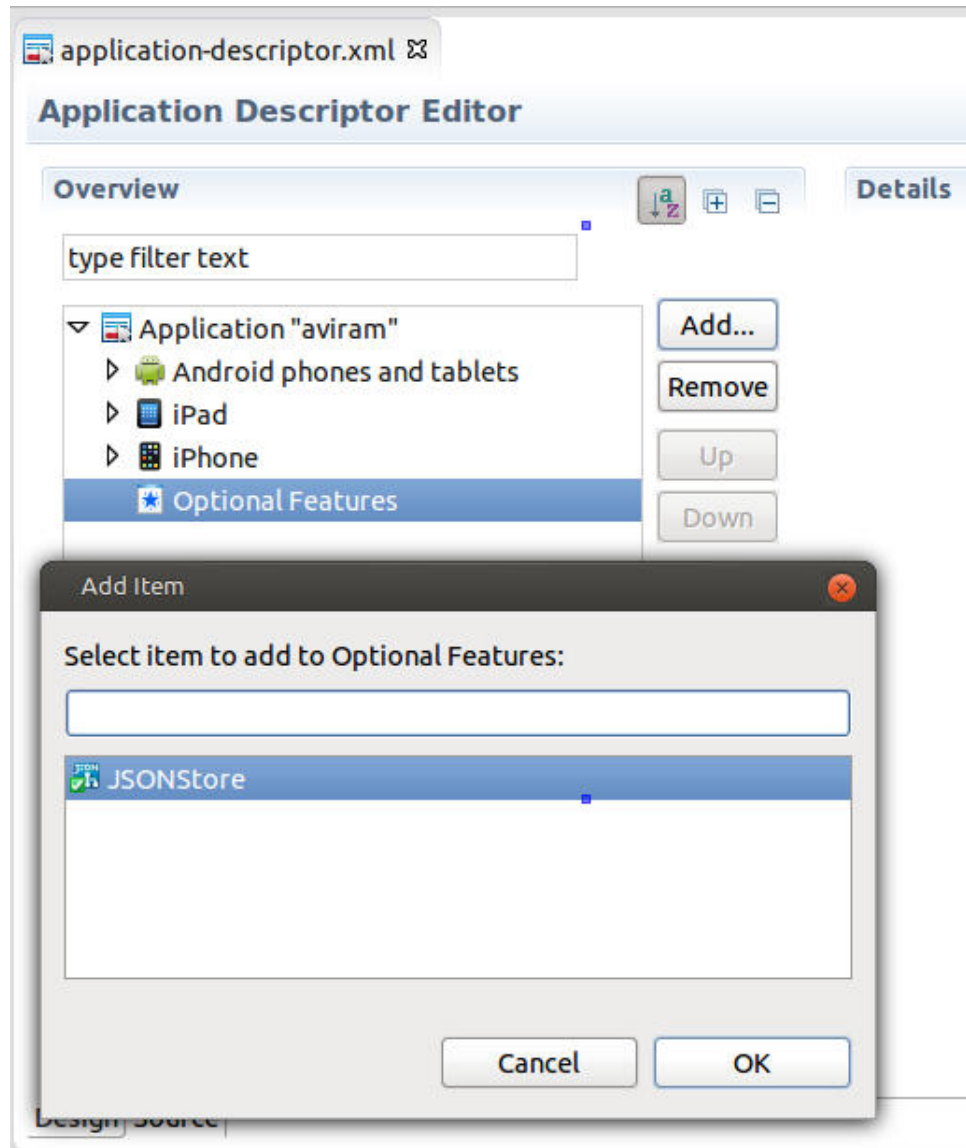
3. Choose the feature that you want to add to the build (in this example, **JSONStore**), and click **OK**.

4. The Application Description Editor now displays **JSONStore** as an attribute of **Optional Features**, along with **Details** about the feature in the right panel:
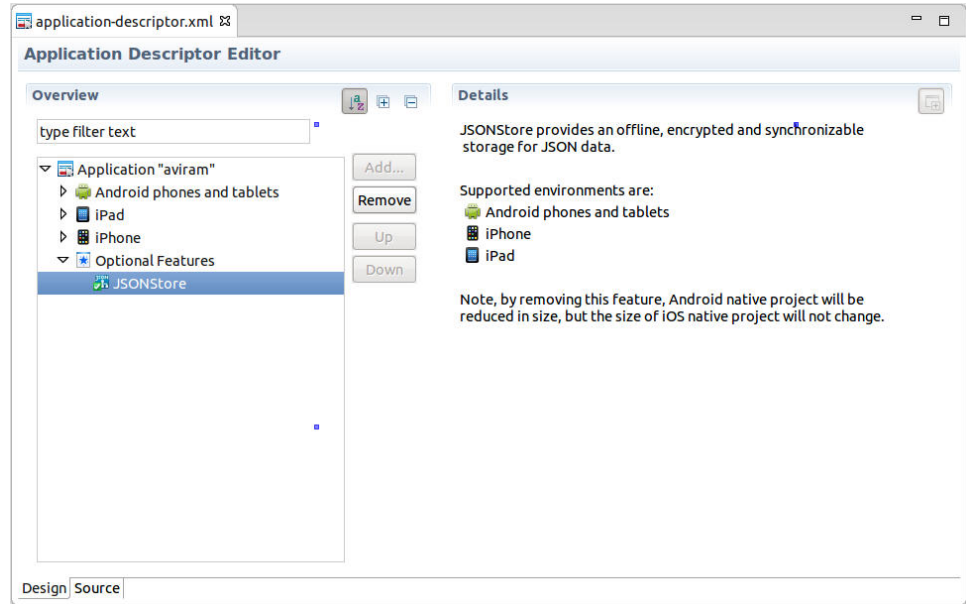
5. To remove a feature, select it in the left panel and click **Remove**.

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 Foundation, you can control the Cache Manifest in web environments (Desktop Browser and Mobile Web). The name of the Cache Manifest file is `worklight.manifest`. This file is located in the folder for each of these types of environments.

You can now view (and edit) the contents of this file in Worklight Studio, as shown in the following screen capture:
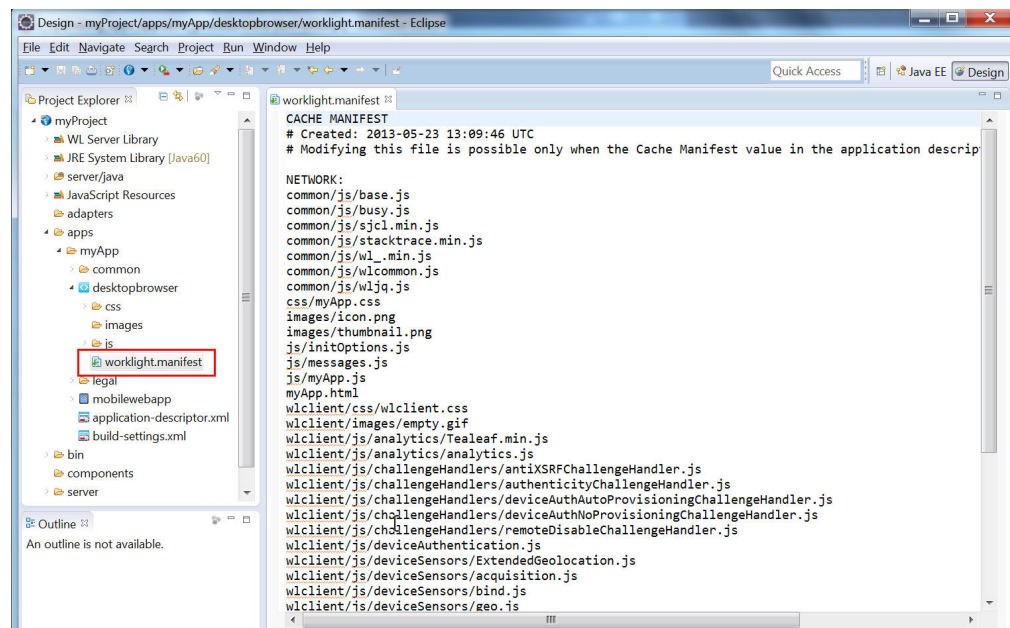
A new attribute now exists in the Application Descriptor (`application-descriptor.xml`) for Desktop Browser and Mobile Web elements. The current setting of this attribute, called `<cacheManifest>`, can be easily viewed, as shown in the following screen capture:

```
<!-- Attribute "id" must be identical to application folder name -->
⊖<application xmlns="http://www.worklight.com/application-descriptor" id="lala" platformVersion=
    <displayName>222</displayName>
    <description>lala22</description>
⊖    <author>
        <name>application's author</name>
        <email>application author's e-mail</email>
        <homepage>http://mycompany.com</homepage>
        <copyright>Copyright My Company</copyright>
    </author>
    <mainFile>lala.html</mainFile>
    <thumbnailImage>common/images/thumbnail.png</thumbnailImage>
⊖    <iphone bundleId="com.lala" version="1.0">
        <worklightSettings include="true"/>
⊖        <security>
            <encryptWebResources enabled="false"/>
            <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif
        </security>
    </iphone>
    <mobileWebApp cacheManifest=""/>
    <worklightServerRootURL>http://${local.IPAd
</application>
                                              ☰ "generated"
                                              ☰ "no-use"
                                              ☰ "user"
```

For more information about Application Descriptor attributes, see "The application descriptor" on page 430.

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 118. `<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:
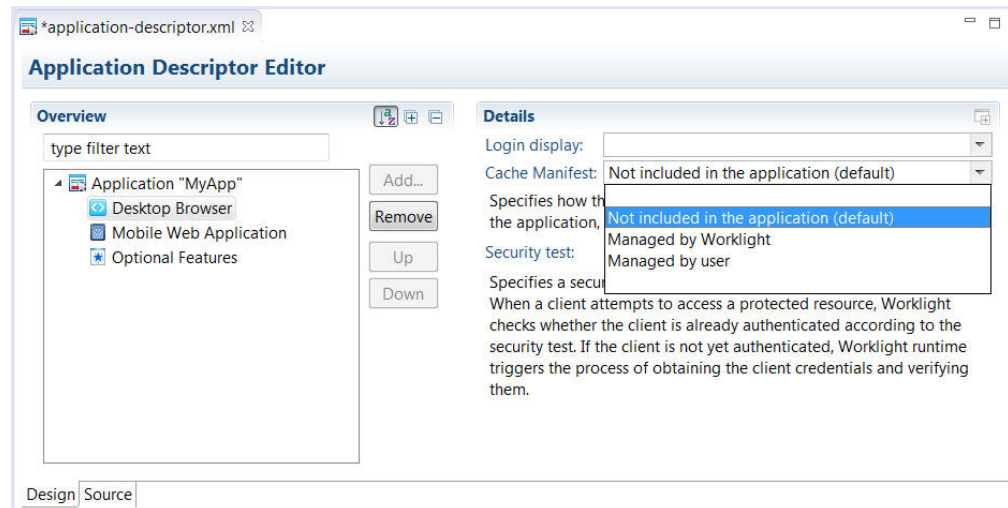
In this view, each of these attribute options is given a description:
- **Not Included in the application (default)** corresponds to `no-use` mode
- **Managed by Worklight** corresponds to `generated` mode
- **Managed by user** corresponds to `user` mode

The only `<cacheManifest>` mode that enables the user to edit the Cache Manifest is **user**. If you attempt to edit the file in any other mode, Worklight Studio displays the following message:
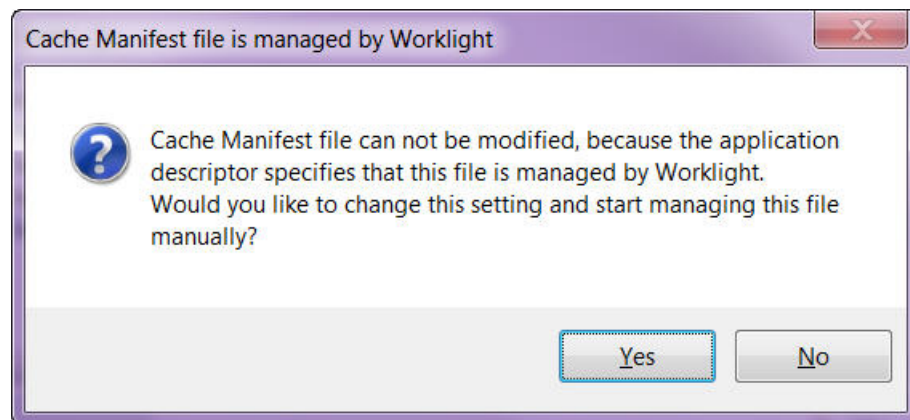
If you click **Yes** on this window, you can change the `<cacheManifest>` mode interactively and then continue to edit the file. You can also change the `<cacheManifest>` mode at any time with Worklight Studio DDE editor.

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.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/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:
*
```

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 119. 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 726.

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 728.

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:
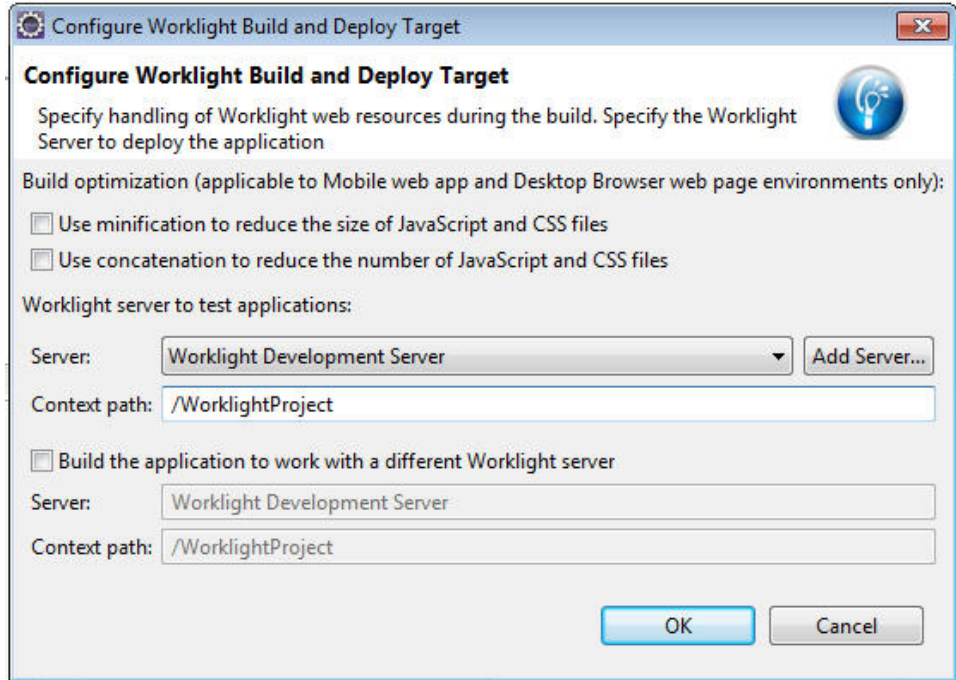
2. In the **Build optimization** area of the dialog, select the check box of the feature or features you want to use when you build this environment.

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:
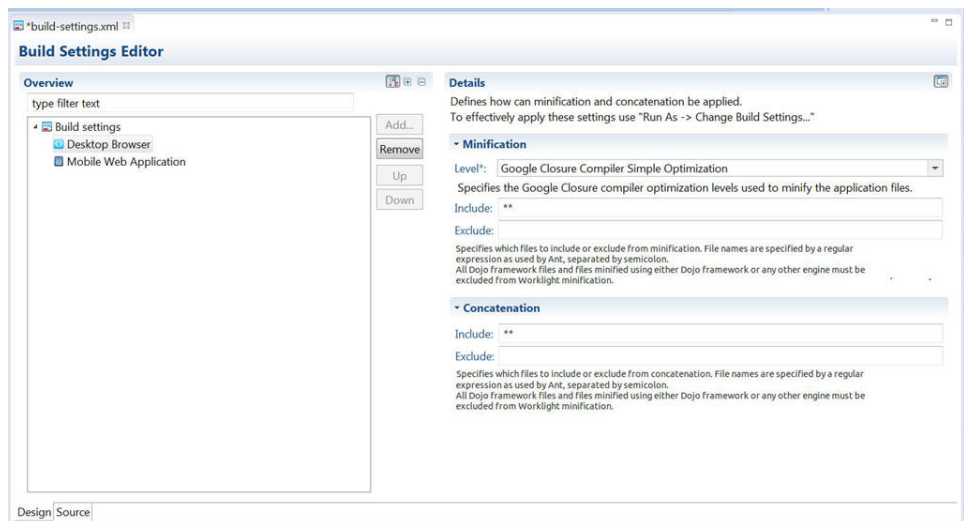
2. To create a configuration for **Concatenation**:

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 "Worklight application build settings" on page 723 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 a Worklight application, as listed in the following table:

Table 120. 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:
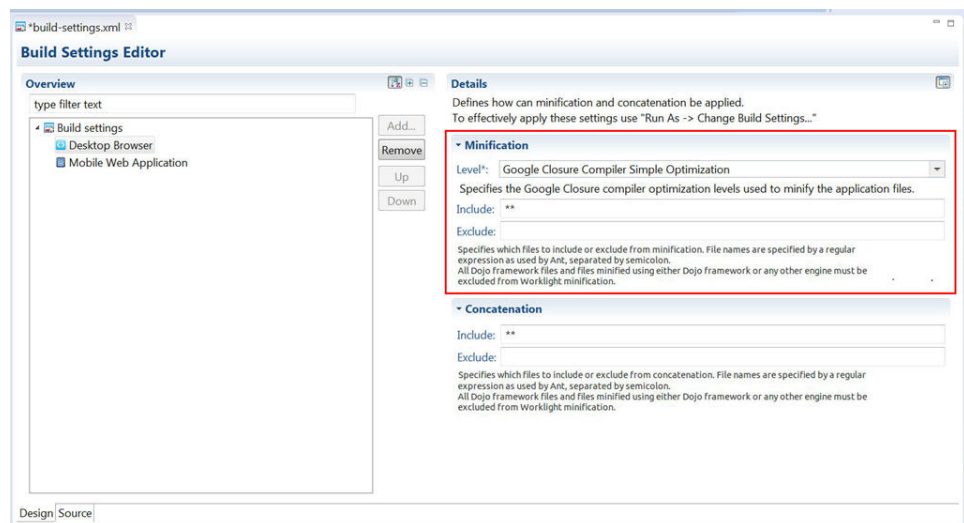
2. Select the wanted minification level from the **Level** field:
   - **None (Default)** specifies the `none` attribute in the table previously mentioned.

- **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 "Worklight application build settings" on page 723.

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:
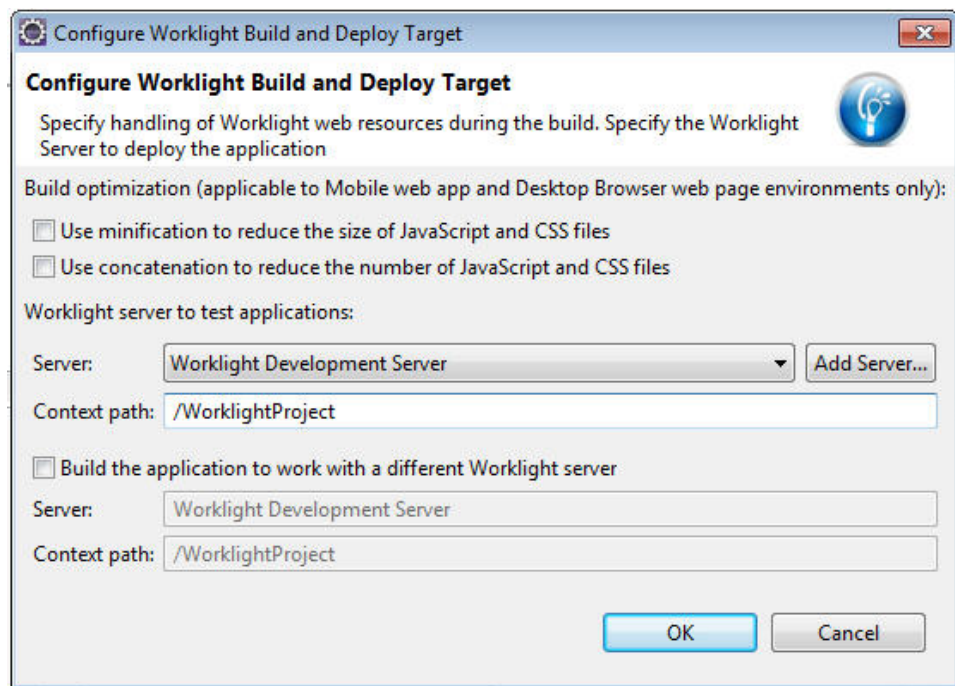
5. In the **Build optimization** area of the dialog, select **Use minification to reduce the size of JavaScript and CSS files**.

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 "Worklight application build settings" on page 723 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:
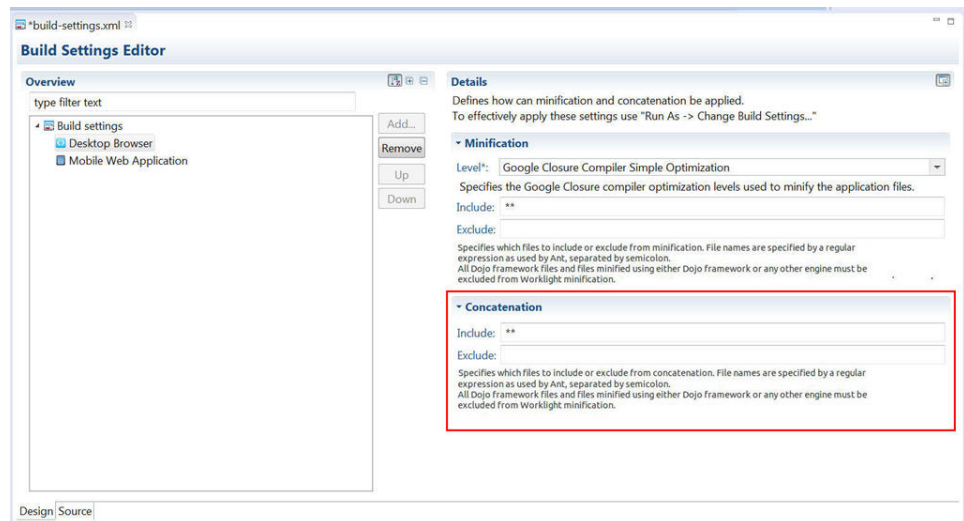
2. Enter the list of files to be concatenated or not concatenated in the **includes**
   and **excludes** fields. When you save, these settings become part of the
   application code.

   Use the Ant syntax that is described in the following "Syntax and examples" on
   page 730 section and in "Worklight application build settings" on page 723.

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:
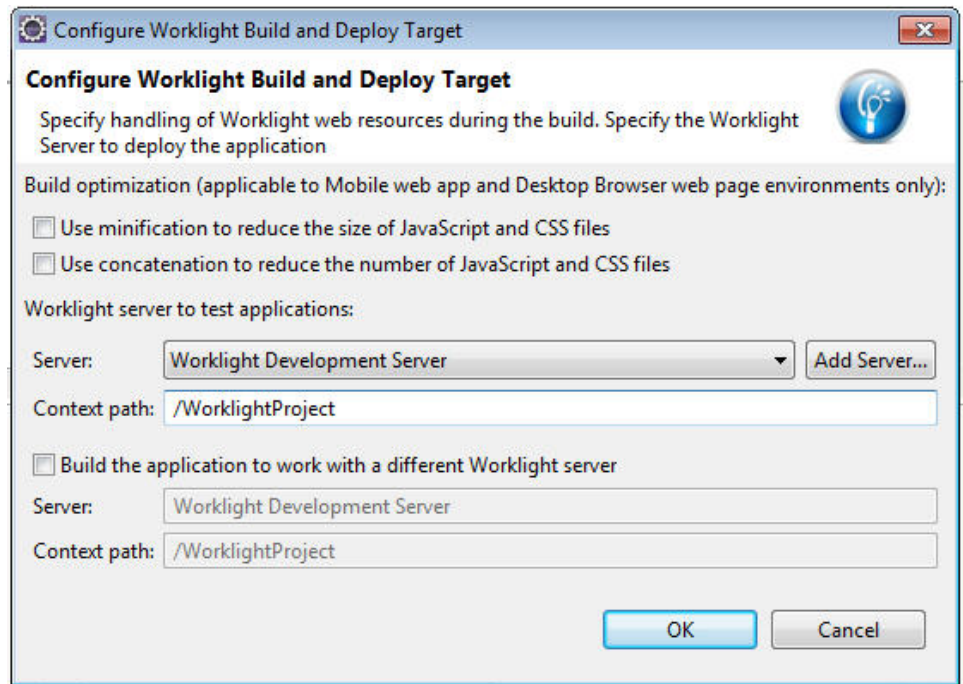
4. In the **Build optimization** area of the dialog, select **Use concatenation to reduce the number of JavaScript and CSS files**.

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 a Worklight HTML file that contains the standard resources that are provided by IBM Worklight Foundation, 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.js`and `myJSFile2.js` were concatenated into the file `wlconcatenatedhead1.js`.
- The file `myJSFile3.js` contains the `async` attribute, and so it was not concatenated into another file.
- The files `myJSFile4.js` and `myJSFile5.js` were concatenated into the file `wlconcatenatedhead2.js`.
- In the body, the files `initOptions.js`, `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.

Optimizing Worklight applications for use over slow networks

If your Worklight applications are meant to run under limited network conditions, you can follow these guidelines to build your applications and improve the performance.

The data transfer rates, in places where the Internet is accessed through GSM, GPRS, or EDGE, might be a few hundred of kilobits per second. The networks with such limited connectivity are considered as *slow networks*.

When the mobile devices, running a Worklight application, access the server over a slow network, it is important to reduce the amount of data transfer between the device and the server. This reduction ensures faster interaction with the user. Even when the amount of data transfer is minimized, it is equally important to keep the user apprised of the estimated time to complete an operation over the network. The following guidelines can be used in Worklight applications development to optimize the mobile user experience over slow networks.

Use compressed response

From IBM Worklight V6.0, it is possible for the Worklight applications to request data in a compressed format in response to the **invokeProcedure** calls. Since the data returned by the Worklight Server is in JSON format, compressing the adapter responses greatly reduces the amount of data that is transferred. The time to complete a response is reduced as well.

The following code snippet demonstrates how to request for a compressed response from the server in a hybrid application.

```
var invocationData = {
  adapter : 'adapter-name',
  procedure : 'procedure-name',
  parameters : [],
  compressResponse : true
}
WL.Client.invokeProcedure(invocationData, options);
```

Configure adapter timeout

Each **invokeProcedure** call that is made by the device is subject to a default timeout of 30 seconds. If your **invokeProcedure** call expects a large amount of data from the adapter over a slow network, you must increase the timeout.

The following code snippet demonstrates how to set the adapter timeout to 60 seconds.

```
var invocationData = {
  adapter : 'adapter-name',
  procedure : 'procedure-name',
  parameters : [],
  compressResponse : true
}
var ONE_MINUTE = 60 * 1000;

var options = {  timeout : ONE_MINUTE,
  onSuccess : sucessCallback,
  onFailure : failureCallback
};

WL.Client.invokeProcedure(invocationData, options);
```

Response time from the adapter

Another factor that influences the **invokeProcedure** timeout on the device is the actual time that is taken by the adapter to respond. If your adapter takes a significant amount of time to fetch data from the backend, then this factor must be considered in the timeout value you set during procedure invocation.

In general, you must ensure

```
(invokeProcedure timeout) >= (Adapter response time) + (Transmission time over the network)
```

Adapt application behavior to timeout

Slow networks are often unpredictable in terms of the speed and reliability. You must be cognizant of this fact and build resilience to timeout within your applications. For example, your applications can try again a timed out operation with a larger timeout value.

The following code snippet demonstrates a typical JSON response from the Worklight Server when a timeout occurs. This response can be accessed in the **onFailure** callback.

```
{
  "invocationContext" : null,
  "errorCode" : "REQUEST_TIMEOUT",
  "errorMsg" : "Request timed out for <REQUEST_URL>. Make sure the host address is available to th
}
```

Use of Application Center

The Application Center provides a mobile client that can be used to download and update applications. Delivering application updates through the Application Center provides the devices with capabilities such as resumable downloads and automatic reload of broken downloads. These features might be useful especially for the devices that install and update applications over slow networks.

Direct update consideration

Successful direct update of applications over slow networks depends on sustained reliability of the mobile network over a large period while the update is being downloaded. Frequent loss of signal, network congestion can lead to direct update failure. You must keep the application footprint as small as possible to minimize chances of direct update failure. IBM Worklight Foundation sends a compressed version of the update file to the client to minimize data charges, and to increase chances of a successful direct update over slow networks. When direct update fails over slow networks, the user must be advised to resume the direct update when a better network, such as WiFi, is available.

The user is prompted with an error message when the direct update fails as a result of network troubles.

```
Update Failed

Failed downloading application update file.

    Exit              Reload
```
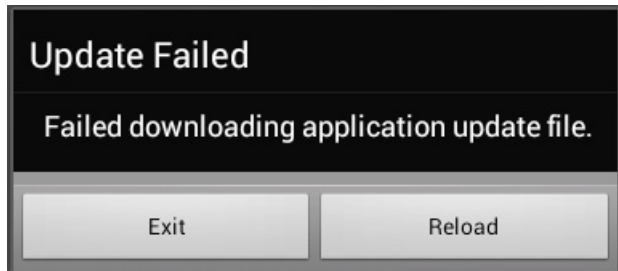
Turn on compression by default

Worklight Server provides a configuration property `compress.response.threshold`. The responses to an `invokeProcedure` call from a device that are above this threshold are automatically compressed by the server. This setting ensures that network performance is optimal even when a client application does not request for compressed data, either due to ignorance or under estimation of the payload size. The default value of the `compress.response.threshold` property is 20480 bytes. For more information, see "Miscellaneous Settings" on page 1177.

Developing the server side of a Worklight application

This collection of topics relates to various aspects of developing the server-side components of a Worklight application.

Overview of 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 Foundation. 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 Foundation, 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 756.

Benefits of 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**: Worklight adapters support read-only and transactional access modes to back-end systems.
- **Security**: 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 Foundation.
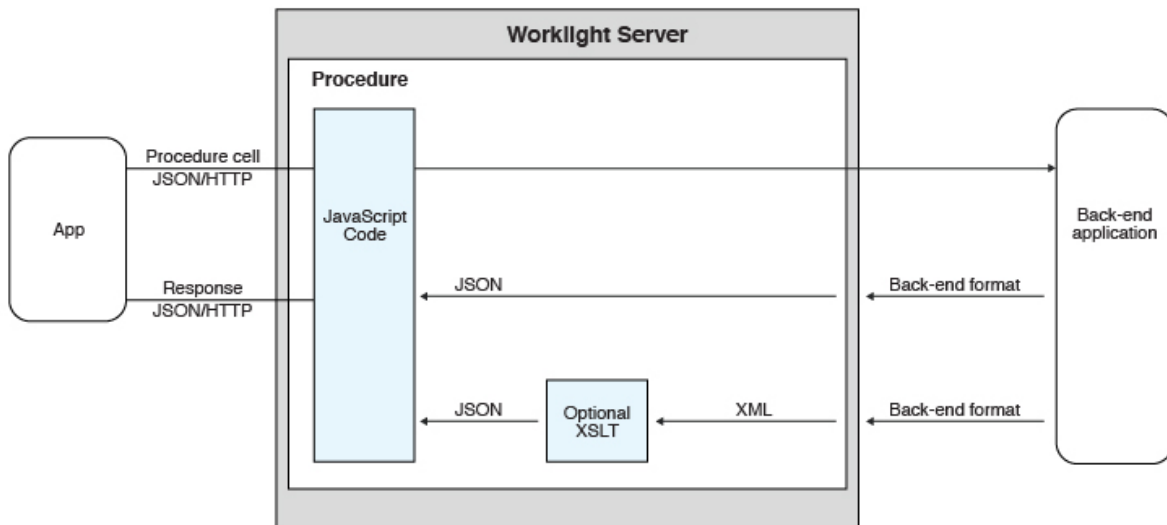
Figure 84. The adapter framework

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 Worklight Server keeps the data intact.

- If this format is not JSON, the Worklight Server automatically converts it to JSON. Alternatively, you can provide an XSL transformation to convert the data to JSON. In this case, the returned content type from the back-end must be XML. Then, you can use an XSLT file to filter the data.

4. The JavaScript implementation of the procedure receives the JSON data, performs any additional processing, and returns it to the calling app.

   -

      Note: Writing an adapter that pulls large amounts of data and transfers it to the client application is discouraged because the data must be processed twice: once at the adapter and once again at the client application. To reduce the possibility of an out-of-memory error, limit the amount of data to less than 10 MB.

   - 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

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 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 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 1176.
- 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 743.
- If you use WebSphere Application Server, you can take benefit of the WebSphere SSL configuration as described in WebSphere Application Server SSL configuration when using HTTP adapters.

Note: 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 Worklight SQL adapter to execute parameterized SQL queries and stored procedures that retrieve or update data in the database.

The Cast Iron adapter

The 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 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 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 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.

The `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 Foundation 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 121. `<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.

Table 121. <adapter> element attributes (continued)

Attribute	Description
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 122. <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 "The `connectivity` element of the adapter XML file" on page 741.

Table 122. <adapter> element subelements (continued)

Subelement	Description
<procedure>	Mandatory. Defines a process for accessing a service exposed by a back-end application. Occurs once for each procedure exposed by the adapter. For more information, see "The procedure element of the adapter XML file" on page 742.

The 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 123. <connectivity> element subelements

Subelement	Description
<connectionPolicy>	Mandatory. Defines back-end-specific connection properties.
<loadConstraints>	Mandatory. Defines the number of concurrent connections which the Worklight Server can open to the back end.

The 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 743
The structure of the <ConnectionPolicy> element.

"The connectionPolicy element of the SQL adapter" on page 747
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 750
The structure of the connectionPolicy element.

The 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.

Worklight queues incoming service requests from Worklight applications. While the number of concurrent requests is below the maximum, IBM Worklight Foundation 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 Foundation 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 Foundation 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)

The 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 124. <procedure> element attributes

Attribute	Description
name	Mandatory. The name of the procedure. This name must be unique within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter.

Table 124. <procedure> element attributes (continued)

Attribute	Description

connectAs	Optional. Defines how to create a connection to the back end for invoking the retrieve procedure. Valid values are as follows:
	
	    `server`: Default. The connection to the back end is created according to the connection policy defined for the adapter.
	
	    `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>
```

Chapter 9. Developing Worklight applications **743**

```
    <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 125. `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 Foundation V6.2.0, starting with Fix Pack 1, 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 126. `ConnectionPolicy` element subelements

Subelement	Description
protocol	Optional. The URL protocol to use. Possible values are `http` (default) and `https`.
domain	Mandatory. The host address.
port	Optional. The port address. The default value is 80.
sslCertificateAlias	The alias of the adapter private SSL key, which is used by the HTTP adapter key manager to access the correct SSL certificate in the keystore. Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication. The keystore setup process is described in "SSL certificate keystore setup" on page 1176

Table 126. `ConnectionPolicy` element subelements (continued)

Subelement	Description
sslCertificatePassword	The password of the adapter private SSL key, which is used by the HTTP adapter key manager to access the correct SSL certificate in the keystore. Optional for regular HTTP authentication and simple SSL authentication. Mandatory for mutual SSL authentication. The keystore setup process is described in "SSL certificate keystore setup" on page 1176
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 746.

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 745.

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: This method is primarily intended for development mode. In production mode, it is recommended 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 127. `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 128. `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 a Worklight project in production by using JNDI environment entries" on page 1182.

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 1121.

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 129. `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 130. `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 a Worklight project in production by using JNDI environment entries" on page 1182.

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 131. `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 132. `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 133. `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 134. `connectionPolicy` element subelements

Subelement	Description
namingConnection	Optional. Describes how to connect to an external JNDI repository. Only used if the JNDI objects are not stored in the JEE server that the adapter is deployed in. See "The `namingConnection` element of the JMS adapter."
jmsConnection	Mandatory. Describes the connection factory and optional security details used to connect to the messaging system. See "The `jmsConnection` element of the JMS adapter" on page 751.

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 ServerWebSphere Application 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 135. `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 136. `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.

Table 136. connectionPolicy element subelements (continued)

Subelement	Description
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 745. The SAP Netweaver Gateway adapter shares the same authentication configuration stanza with the HTTP adapter except that `<serverIdentity>` requires one additional `<client>` tag.

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 1176.
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 1176.

Table 136. `connectionPolicy` element subelements (continued)

Subelement	Description
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 746. The SAP Netweaver Gateway adapter shares the same proxy configuration stanza with the HTTP adapter.

Creating a Worklight adapter

Follow these instructions to create a Worklight project and configure a new 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 do not already have an existing Worklight project. If you set up Worklight shortcuts, right-click anywhere within the Eclipse Project Explorer view and click **New** > **Worklight Project**. Otherwise, click **New** > **Other**, then select **Worklight** > **Worklight Project** from the list of wizards and click **Next**.
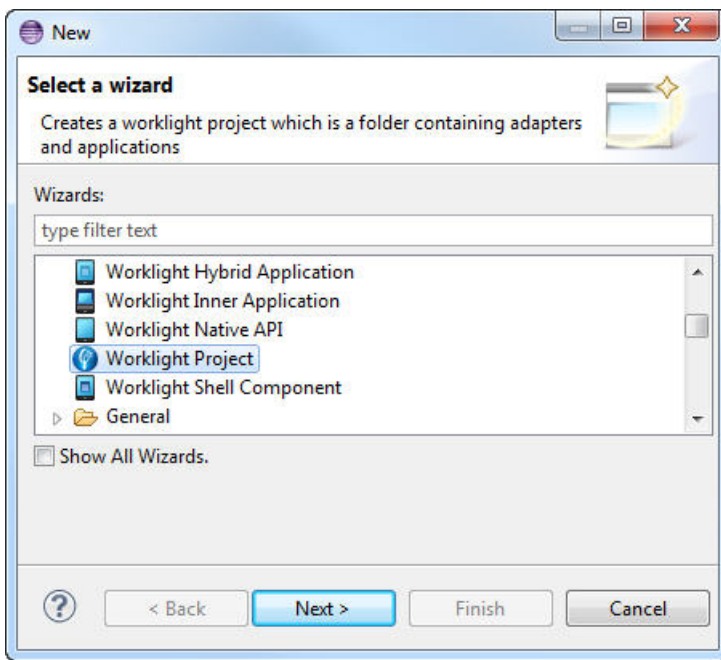
Figure 85. Creating a Worklight project from the wizard.

2. In the **New Worklight Project** wizard, specify a name for the project and click **Finish**.

3. If you set up Worklight shortcuts, right-click the 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**.
   The **New Adapter** wizard opens.
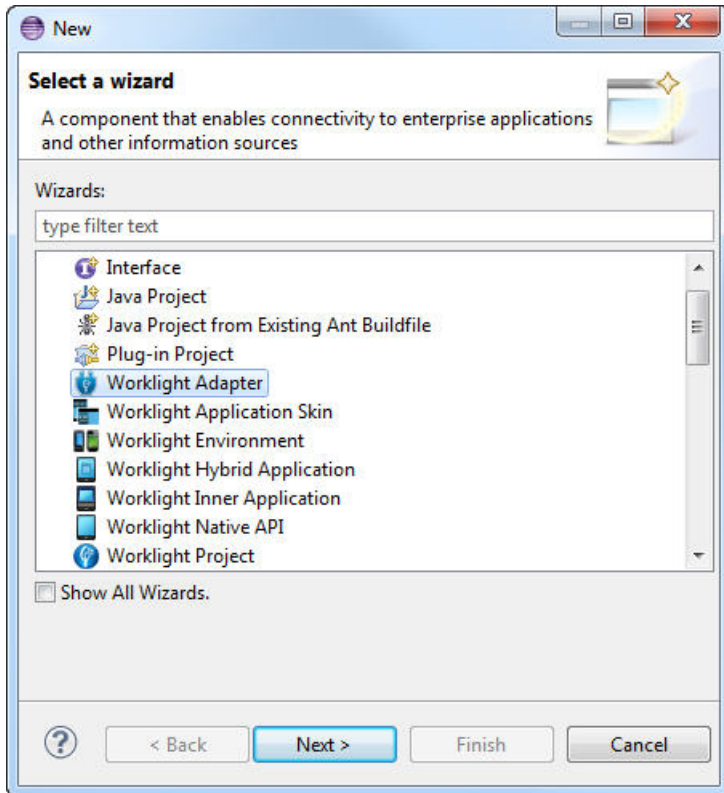
Figure 86. Configuring a new Worklight adapter.

4. Select the required adapter type from the **Adapter type** list and enter a name for the adapter in the **Adapter name** field.
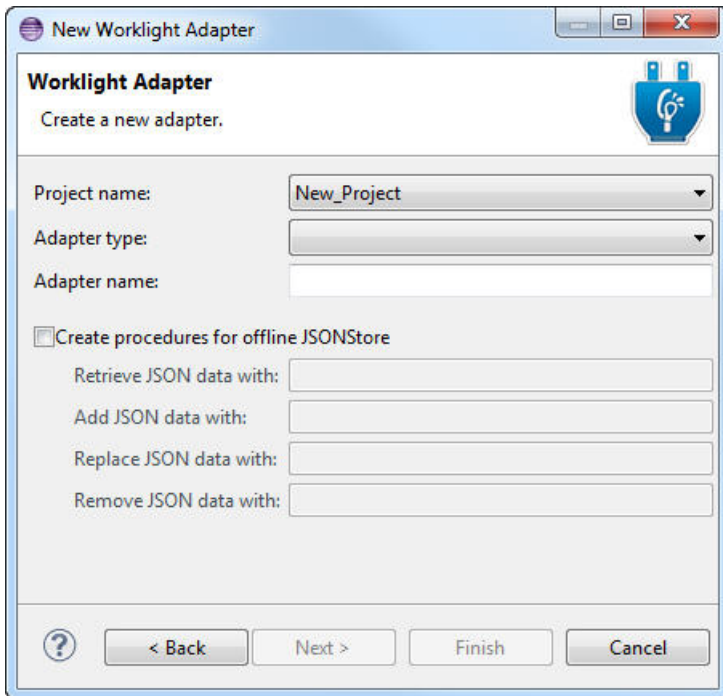
Figure 87. Selecting an adapter type.

5. Optional:
   - Select **Create procedures for offline JSONStore** to include four place holder procedures in the adapter template: a procedure that gets data, a procedure that adds data, a procedure that replaces data, and a procedure that removes data. These procedures are designed to help you develop a JSONStore-enabled application that communicates with a back end.
   - Select **Create procedures for USSD enablement** to generate sample procedures for USSD in the adapter js file.
6. Click **Finish**.

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 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 that you want to use, as described in the following sections:
   - **WSDL** service type:
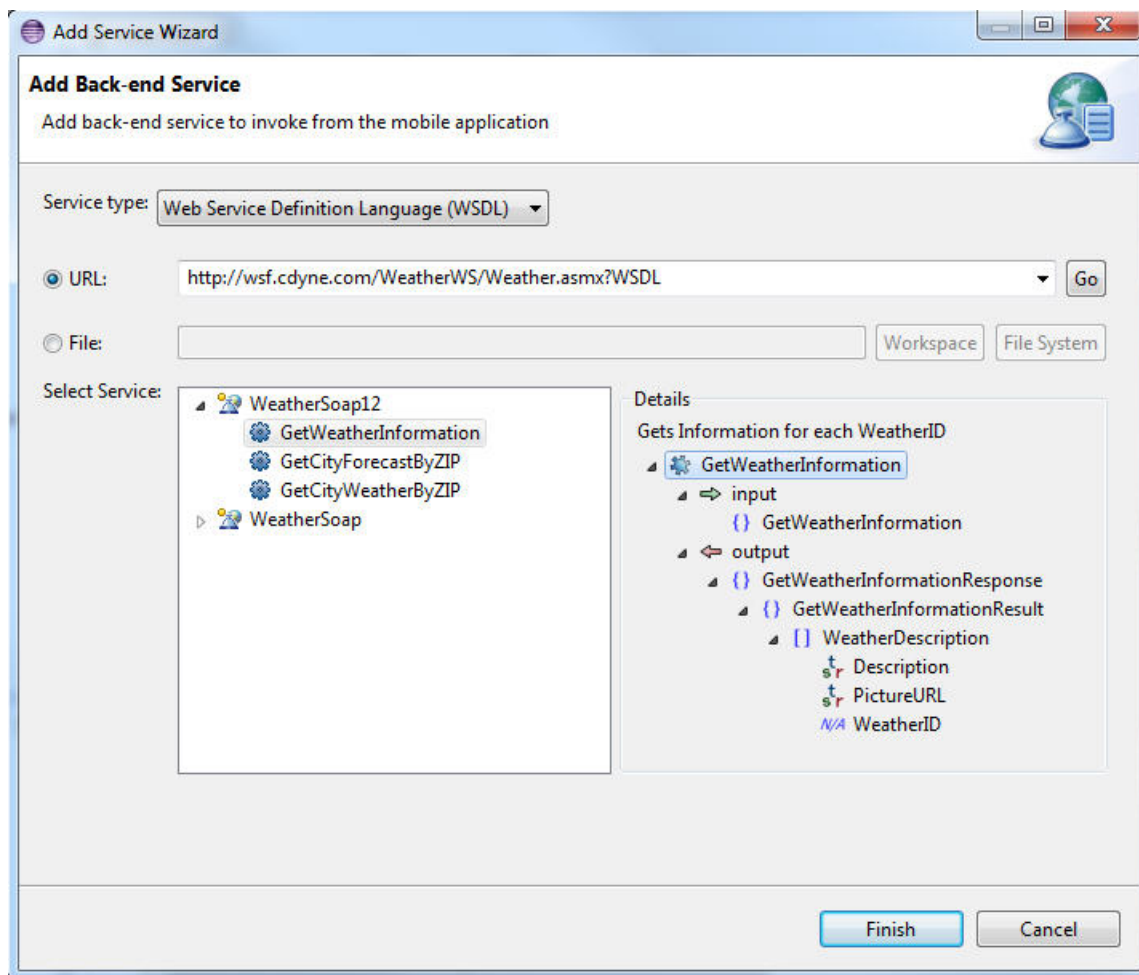
Figure 88. Adding a web service

   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:
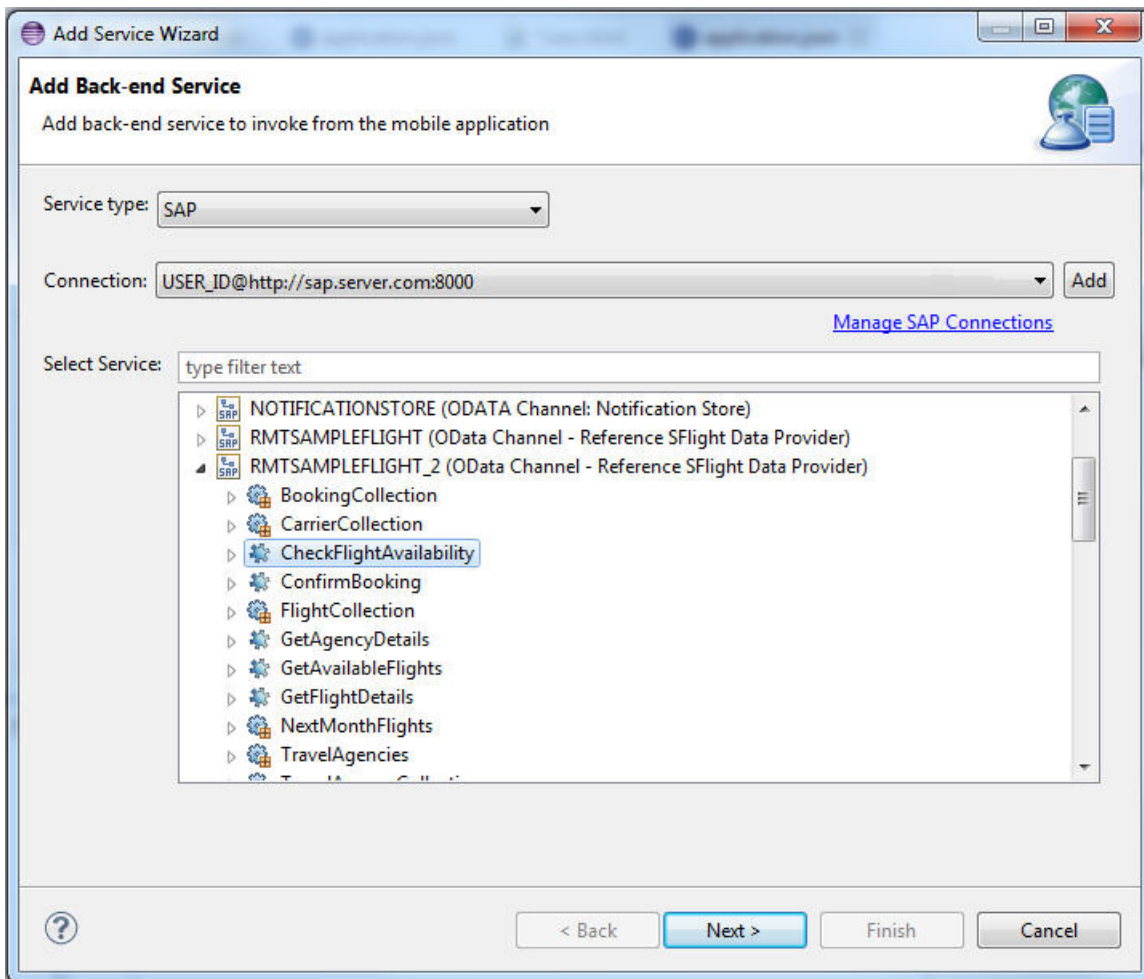
Figure 89. Adding a service exposed by SAP

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 "Creating hybrid applications with IBM Worklight Application Framework" on page 559).

  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 WSDL-based services, a sample payload of each service is available from the Properties dialog box of the service. You can reuse this sample payload in other adapters that are created without IBM Worklight Application Framework. To use the sample payload of a service:

  1. Right-click the service name in the `services` folder of your project in the Eclipse Project Explorer view, and click **Properties**.

  2. Copy the text from the **Sample Procedure Parameter** field.

For more information about:

- invoking the generated SOAP adapters procedure, see "Invocation of generated SOAP adapters" on page 760.

- the content of generated SAP adapters, see "The `root` element of the SAP Netweaver Gateway adapter XML file" on page 752 and "The `connectionPolicy` element of the SAP Netweaver Gateway adapter" on page 752.
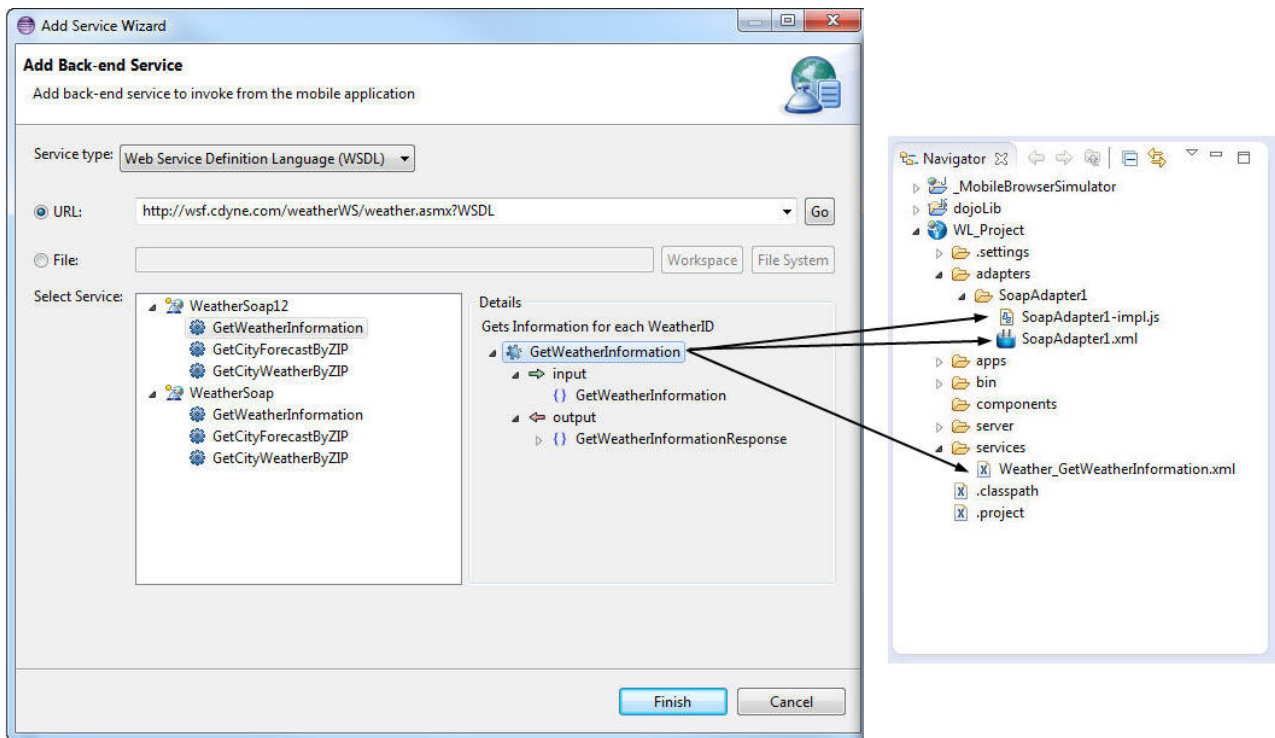
Figure 90. Files generated from the services discovery

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 Worklight application in the same way as you invoke other 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>
   ```

   Note: Since IBM Worklight Foundation V6.2.0, if the names of the elements are unique, the generated SOAP adapters no longer require the use of namespace prefixes on all the fields in the payload.

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 Worklight invocation service: `http(s)://<server>:<port>/<Context>/invoke`.

The following parameters are required:

Table 137. 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 "Worklight security framework" on page 860 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 874 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 Foundation 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

Encode a SOAP XML envelope within a request body when you need to invoke a SOAP-based service in an HTTP adapter.

About this task

Important: This workaround is only for WebSphere Application Server.

Procedure

1. 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. 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` by using the key in the specified **keystoreAlias**, and inserting the digital signature into the input document.

   To use WL.Server.signSoapMessage() API when running IBM Worklight Foundation on IBM WebSphere Application Server, you might need to add a JVM argument that instructs the application server to use a specific **SOAPMessageFactory** implementation instead of a default one.

2. To do this, go to **Application servers** > *{server_name}* > **Process definition** > **Java Virtual Machine** and provide the following argument under **Generic JVM arguments**.

   Type in the code phrase exactly as it is presented here:
   ```
   -Djavax.xml.soap.MessageFactory=com.sun.xml.internal.messaging
   .saaj.soap.ver1_1.SOAPMessageFactory1_1Impl
   ```

3. Restart the JVM.

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>;
```

Backend responses in adapters

Understanding the logic of invocation results both on the client side and inside adapters helps you handle different failure scenarios.

HTTP adapter flow

For a general description of an adapter flow, see "Overview of Worklight adapters" on page 734. The following sections explain how to handle backend responses in the case of an HTTP adapter. A typical HTTP adapter flow might involve the following sequence of events:

1. The client (that is, the mobile app) uses the `invokeProcedure` method of the WL.Client class to invoke one of the adapter's procedures from the Worklight Server.
2. The adapter then uses the `invokeHttp` method of the WL.Server class to call the backend service.
3. The adapter procedure processes the data from the backend and returns a JSON object to the client.
4. The client calls its `onSuccess` handler to process the data received by the adapter.

Responses from the invoke procedure

The adapter flow starts with a WL.Client class `invokeProcedure` call, which supports `onSuccess` and `onFailure` handlers. Both handlers receive an object, which is a standard JSON object. The following table describes some of its properties:

Table 138. Properties of the object received following the invoke procedure

Property	Description
isSuccessful	Whether the procedure call is successful. **Note:** The text following the table explains the circumstances when a request is considered to be successful.
status	HTTP status code from the procedure call. This is not the HTTP code from the backend service, only from the connection with the Worklight Server.
errorCode	A possible error code if the call is not successful.
errorMsg	A possible error message if the call is not successful.
invocationContext	An optional object that is sent in the procedure call and is returned as-is.
invocationResult	JSON object that is returned by your procedure call. This object may be augmented with additional data such as session information.

Which handler is called depends on the value of the `isSuccessful` property in the invocation result:

- If `isSuccessful` is set to `true`, `onSuccess` is called.
- If `isSuccessful` is set to `false`, `onFailure` is called.

As long as your adapter returns something, the procedure invocation is considered successful and so the `isSuccessful` property is set to `true`. The `isSuccessful` property is set to `false` under the following circumstances:

- When calling a procedure that does not exist.
- When calling an adapter that does not exist.
- When the Worklight Server is unresponsive (for example, due to a bad hostname or because the Worklight Server is currently unavailable).
- When the invocation times out (you can set a timeout value as one of the `invokeProcedure` options).
- When the adapter throws an exception.
- When the code in the procedure specifically overwrites the `onSuccess` property.

The `isSuccessful` property is set to `false` if there is a connection issue between the client and the adapter; not if there is an error in the backend service. This means, for example, that if your procedure calls a backend service which returns an error (such as a "404" error) but your procedure still returns a valid JSON object, your procedure invocation is still considered to be successful from the perspective of the client. If you simply return the result of `invokeHttp` straight to the client, since you are returning something, `isSuccessful` is `true` by default and `onSuccess` is called. This may or may not be what you want to happen. You need to make sure that your procedure code is capable of handling cases when a backend service returns an error.

Invocations from the adapter to the backend

From your procedure, you call a remote backend service by using the `invokeHttp` method of the WL.Server class. The returned object from this call is a JSON object that represents the result of the HTTP request. If the response is an XHTML or XML tree, it is converted to JSON. For example, if the response is an HTML page, you see a property called "html" (the root HTML tag) with the content tree inside.

The following table describes some of the other properties. Additional arbitrary properties might also be returned by the backend service.

Table 139. Properties of the object received following the invocation from the adapter to the backend

Property	Description
`errors`	Array of errors during the request.
`isSuccessful`	Boolean value summarizing whether the request is successful. **Note:** The text following the table explains the circumstances when a request is considered to be successful.
`responseHeaders`	JSON object representing the different HTTP headers of the response.
`responseTime`	HTTP response time.
`statusCode`	HTTP status code of the remote invocation.

Table 139. Properties of the object received following the invocation from the adapter to the backend (continued)

Property	Description
statusReason	Short text description that explains the status code.
totalTime	Response time plus any additional time for IBM Worklight to complete processing or convert formats.

Similar to the client side, if isSuccessful is set to true, the data that you receive is not necessarily exactly what you expect. It merely indicates that something was returned. You can therefore assume that isSuccessful is true by default. This includes the following cases:

* The remote HTTP server returns an OK status code such as 200.
* The remote HTTP server returns **any valid status code** such as 2XX, 3XX, 4XX. 5XX, and other codes.

The isSuccessful property is set to false under the following circumstances:

* The HTTP host cannot be reached or is invalid.
* The HTTP request has timed out.

Because isSuccessful is set to true by default, you might not receive the data that you want or expect. For example, you might want a "404" error to be treated as a failure whereas IBM Worklight considers it a success. You can use properties such as the statusCode property that is returned in the result of a WL.Server.invokeHttp call (or any other interesting data from the response) to decide if the procedure should be considered successful or not. You can then handle situations that should be considered unsuccessful in one of the following ways:

* Overwrite the isSuccessful property by setting its value to false in your JSON response.
* Consider the request as successful, set some custom flags in your JSON response, and handle the situation in your client's onSuccess handler. You might also want to place a try/catch block around your procedure code and handle any exceptions accordingly. If an exception is thrown, the client will receive an isSuccessful response that is set to false.

In a production environment, returning the result of the invokeHttp call back to the client might not be the ideal value to return at the end of the procedure for the following reasons:

* The meaning of a "successful" request might vary in different cases.
* The backend response might include additional data that should not be forwarded to the client; such as certain response headers, architecture of the backend, or any data that is not relevant to the logic of the app. Instead, consider building a new JSON object with your own data, which might possibly include parts of the original response.

Example

Here is an example of an adapter that receives a "404" error as a result of trying to get data from an invalid URL: www.ibm.com/no-such-place.

adapt.xml

This file can be generated from the Design view in Worklight Studio. The backend hostname is set to www.ibm.com.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="adapt"
  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>adapt</displayName>
  <description>adapt</description>
  <connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
      <protocol>http</protocol>
      <domain>www.ibm.com</domain>
      <port>80</port>
    </connectionPolicy>
    <loadConstraints maxConcurrentConnectionsPerNode="2" />
  </connectivity>
  <procedure name="test"/>
</wl:adapter>
```

adapt-impl.js

This is the implementation of the procedure. It calls a remote URL and generates a response. If the HTTP status code is anything other than 200, the isSuccessful property is set to false.

```javascript
function test(){

  var input = {
    method : 'get',
    path : 'no-such-place' //Replace this with a valid path to see success
  };
  var backendResponse = WL.Server.invokeHttp(input);
  var procedureResponse = {};

  if(backendResponse.isSuccessful && backendResponse.statusCode == 200){
    //For simplicity, considering only 200 as valid
    //Do something interesting with the data
    procedureResponse.interestingData = backendResponse.html.head.title;
  }
  else{
    procedureResponse.isSuccessful = false; //Overwrite to failure
  }

  return procedureResponse;
}
```

main.js

The client app invokes the procedure. If the request is successful, the app logic continues. If the request is not successful, an error message is displayed.

```javascript
WL.Client.invokeProcedure({
  adapter : 'adapt',
  procedure : 'test'
}, {
  onSuccess : function(result) {
    //Do something interesting with resulting JSON
    $('#someDiv').html(result.invocationResult.interestingData);
  },
  onFailure: function(result){
    WL.SimpleDialog.show("Error","The service is temporarily not available. Please try aga
  }
});
```

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.
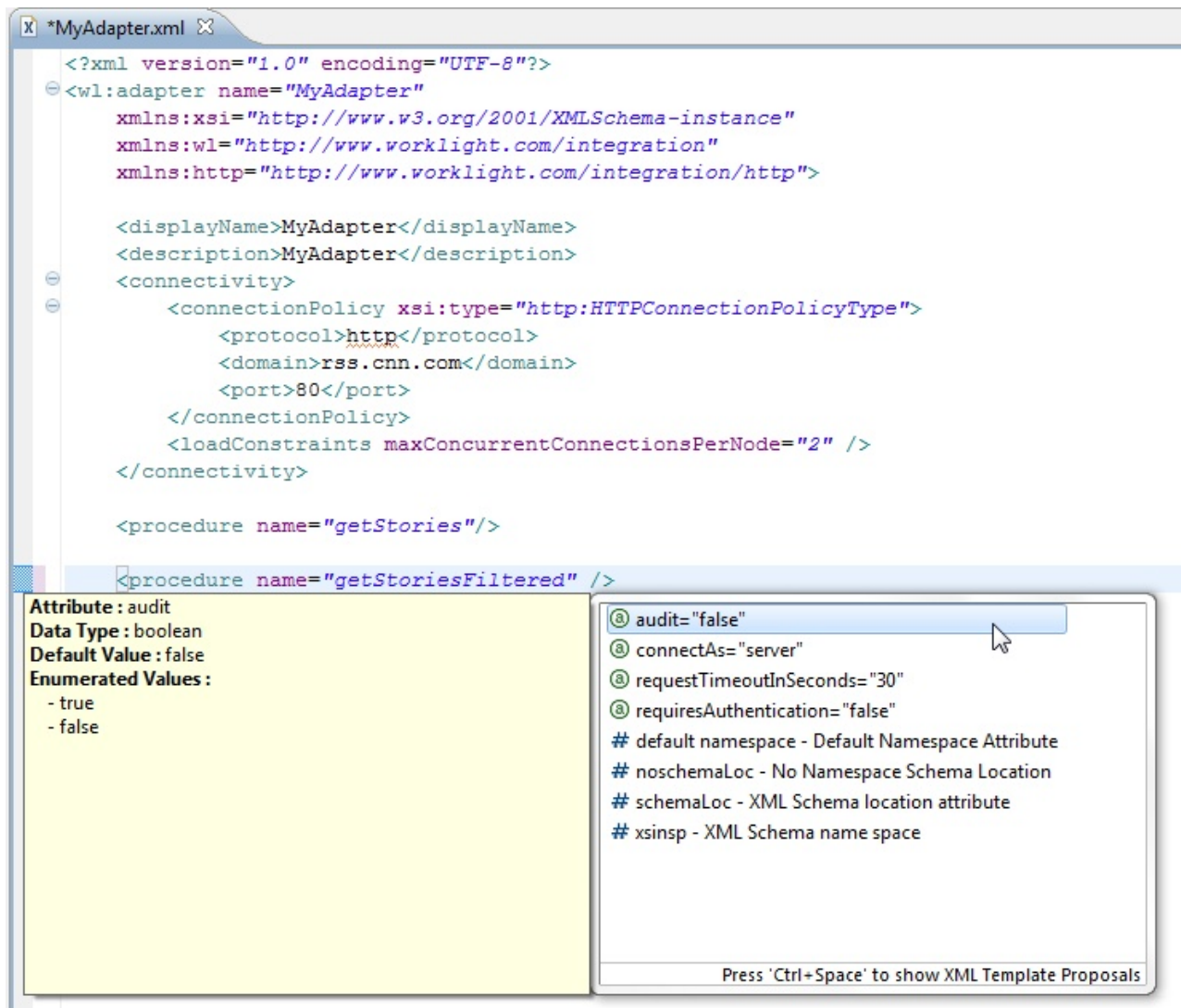
Figure 91. Adapter configuration through the auto-complete feature.

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 Worklight adapter constraints. For example, if a procedure is a JavaScript procedure, then field mapping is not permitted.

XML/JavaScript validation
> Worklight Studio provides validation between XML and JavaScript. It verifies that each declared JavaScript procedure has a corresponding procedure in the adapter JavaScript file with the appropriate signature (that is, input parameters and return values).

Quick fix

The Worklight Studio provides Quick Fix options for adapter configuration errors.

Whenever an error is detected, the error console displays the offending code. To activate the Quick Fix window, right-click the error in the console and select **Quick Fix**. Alternatively, press Ctrl+1.
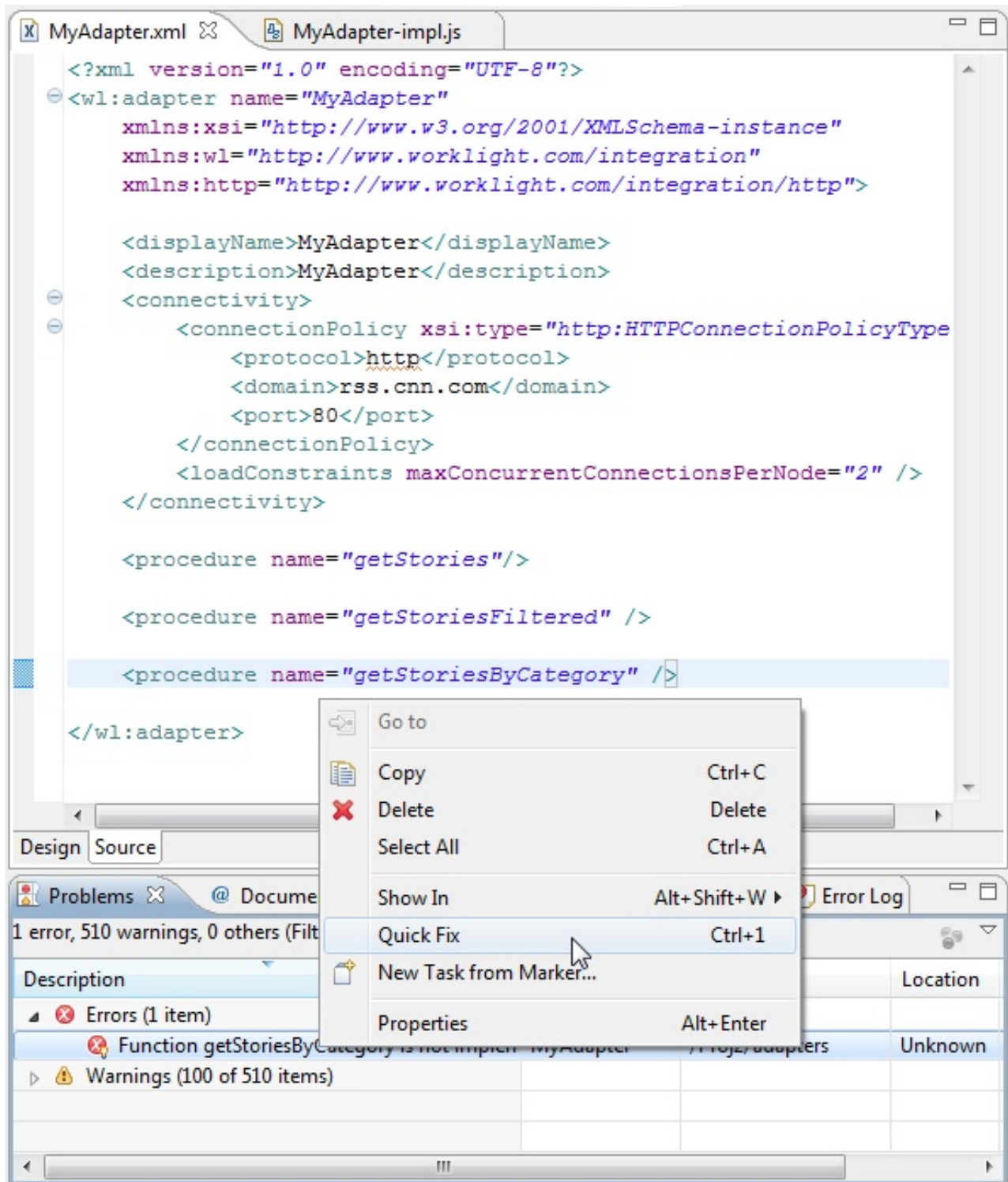
Figure 92. Quick Fix options for adapter configuration problems.
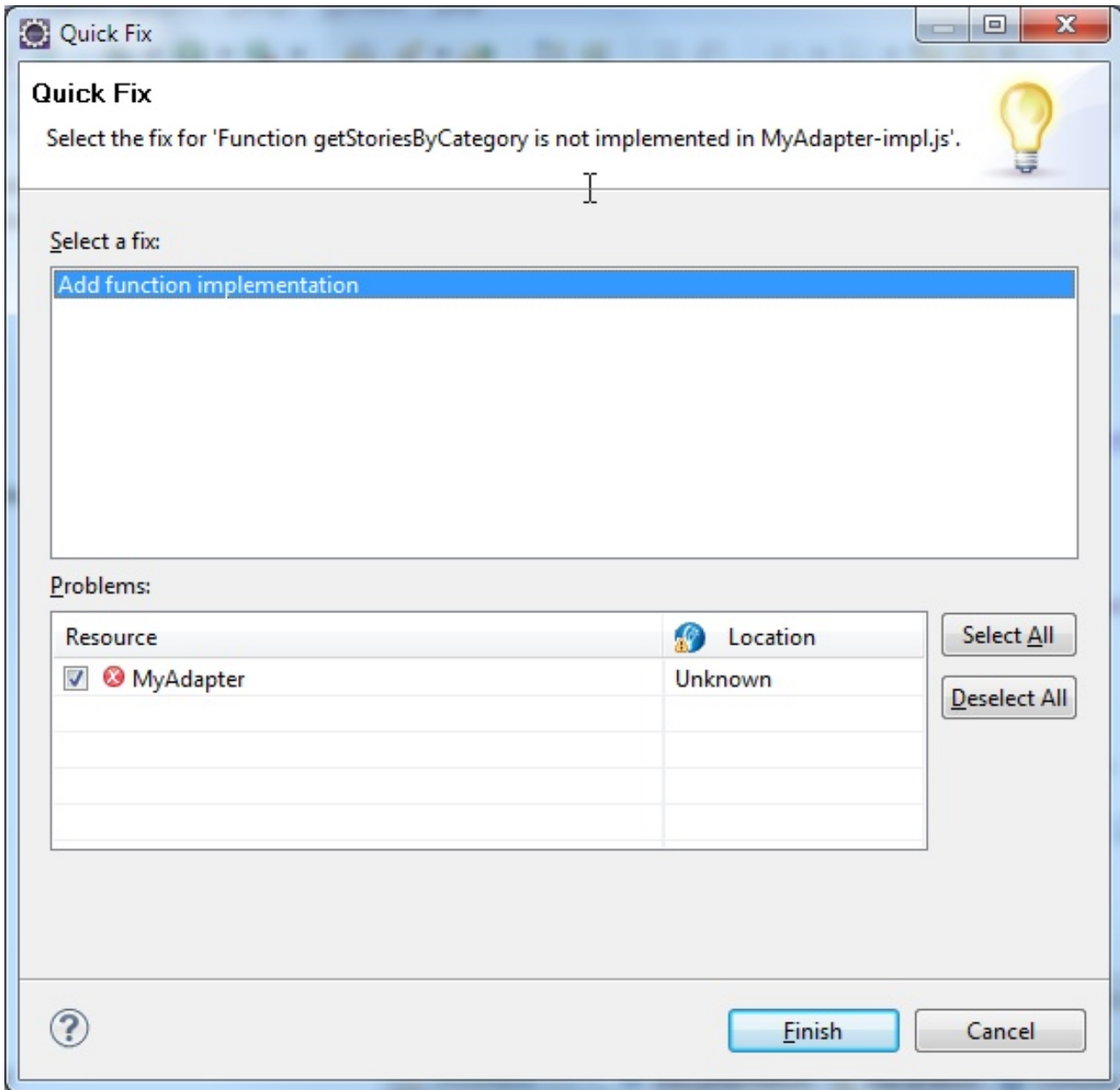
Figure 93. Quick Fix option for missing JavaScript functions.

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.
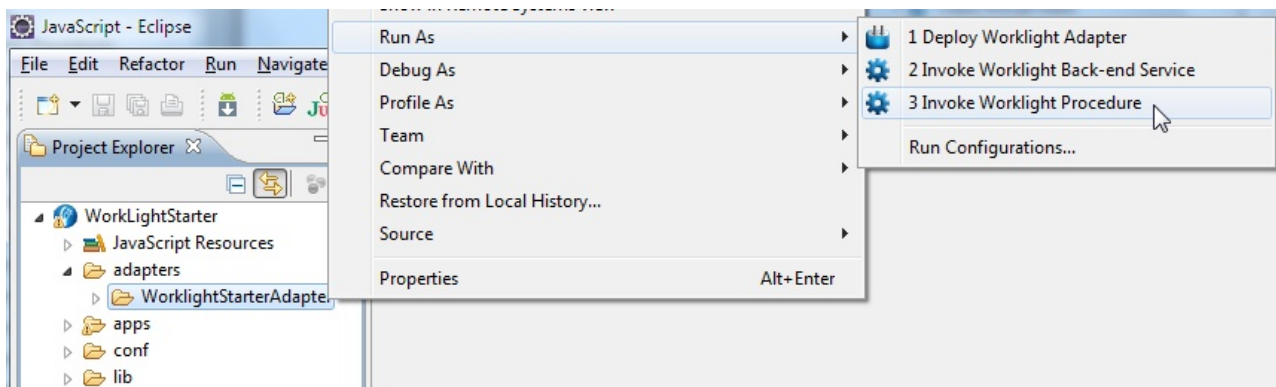
Figure 94. Invoking Worklight procedures.

In the dialog box, provide a comma-separated list of procedure parameters.
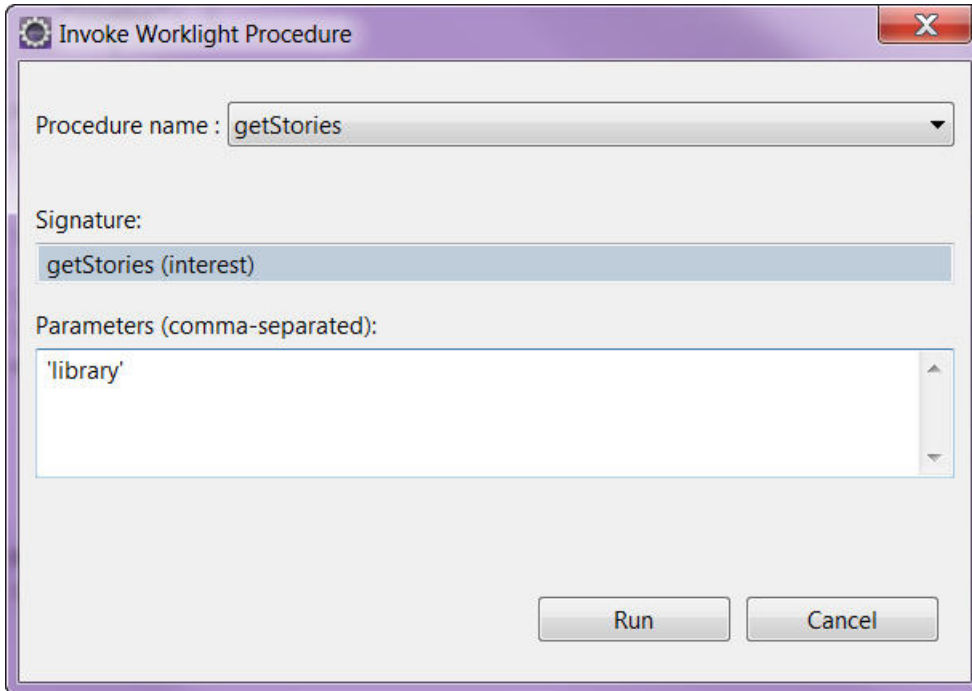
Figure 95. Launching the procedure.

Invoking a back-end service

You can use Worklight Studio to invoke a back-end service and receive the data retrieved by the service.

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**.
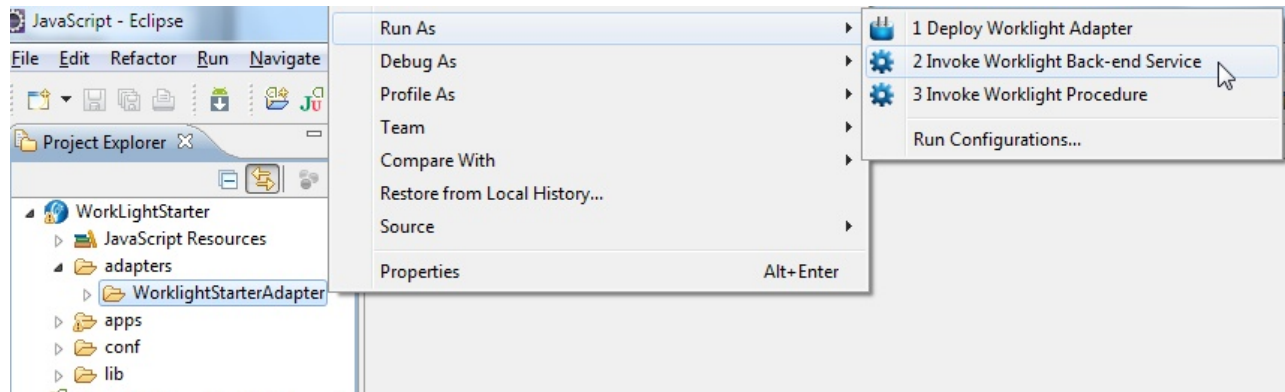
Figure 96. Invoking a Worklight back-end service

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
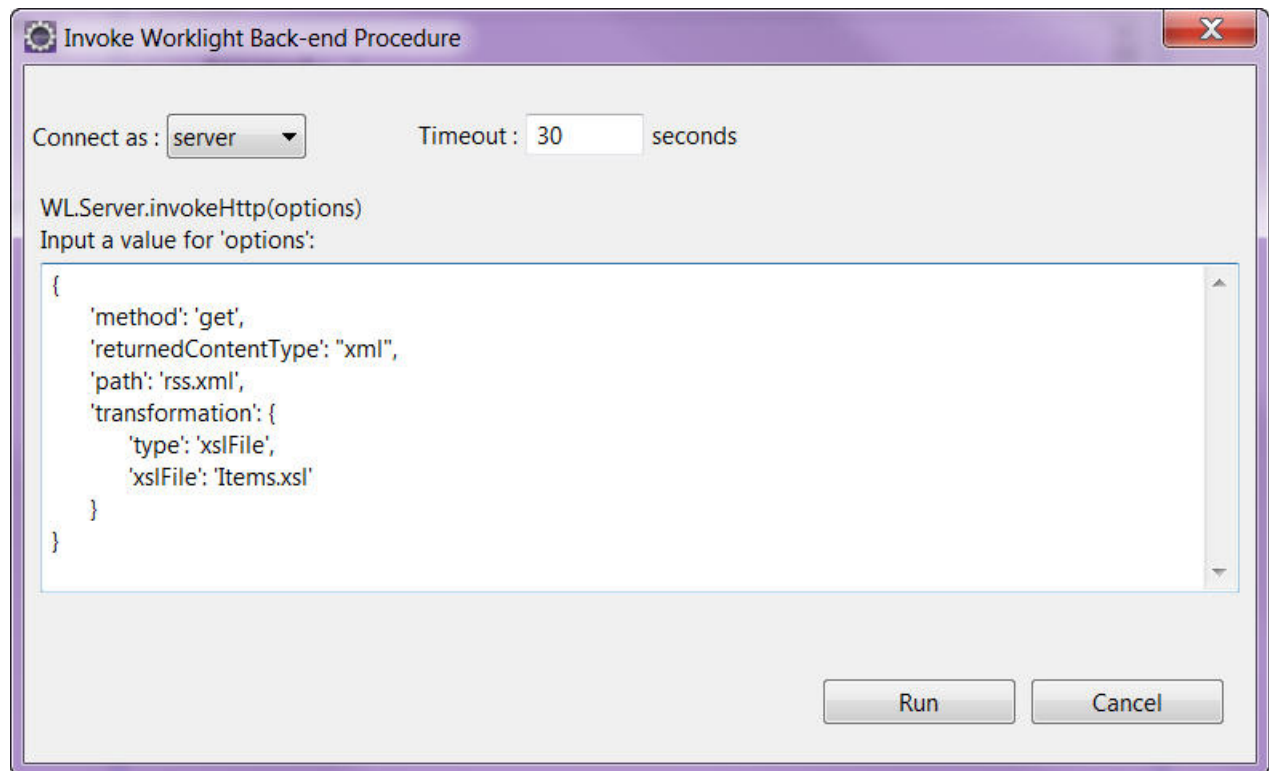
Figure 97. Invocation parameters.

format, and the XSL transformation (if defined) that was used to convert the XML to JSON.

3. Optional: Change the XSL transformation by editing it in the edit box, then click **Apply XSL** to regenerate the JSON format.

Figure 98. 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 Worklight Server.

Procedure

Right-click the adapter folder and select **Run As** > **Deploy Worklight Adapter**.
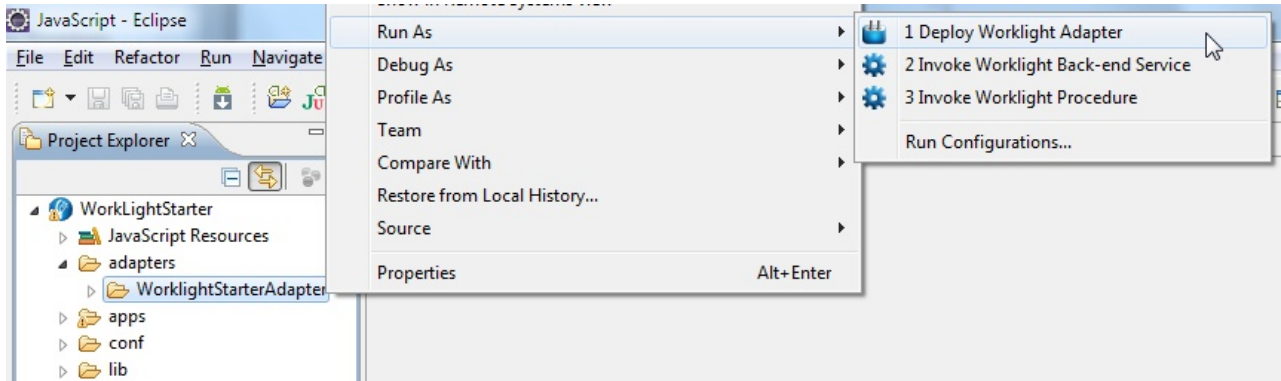
Figure 99. Deploying a Worklight adapter.

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 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 445.

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 a 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']
```

USSD Support

Unstructured Supplementary Service Data (USSD) is a communication technology that is used by GSM cellular telephones to send text messages between a mobile phone and an application program in the network.

USSD establishes a real-time session between the mobile phone and the application that handles the service.

IBM Worklight Foundation uses the HTTP/HTTPS protocol to communicate with the USSD gateway, which is a third-party entity. The USSD gateway routes USSD messages to the Worklight Server. Adapter procedures need to be defined to process these requests and send back a response. You need to define USSD event handler to route the requests to the adapter procedure that handles those requests.

Note: For more information, see the WL.Server.createUSSDEventHandler and WL.Server.createUSSDResponse APIs in WL.Server.

Here is a sample flow for USSD:

1. A mobile user enters a USSD short code, such as *123#.
2. The request is forwarded to a USSD gateway.
3. The gateway maps the short code to a known URL provided by Worklight, creates the USSD session, and forwards the request to the URL.
4. A Worklight adapter with the matching filter receives the request and responds to the gateway request with the configurable USSD menu/simple text.
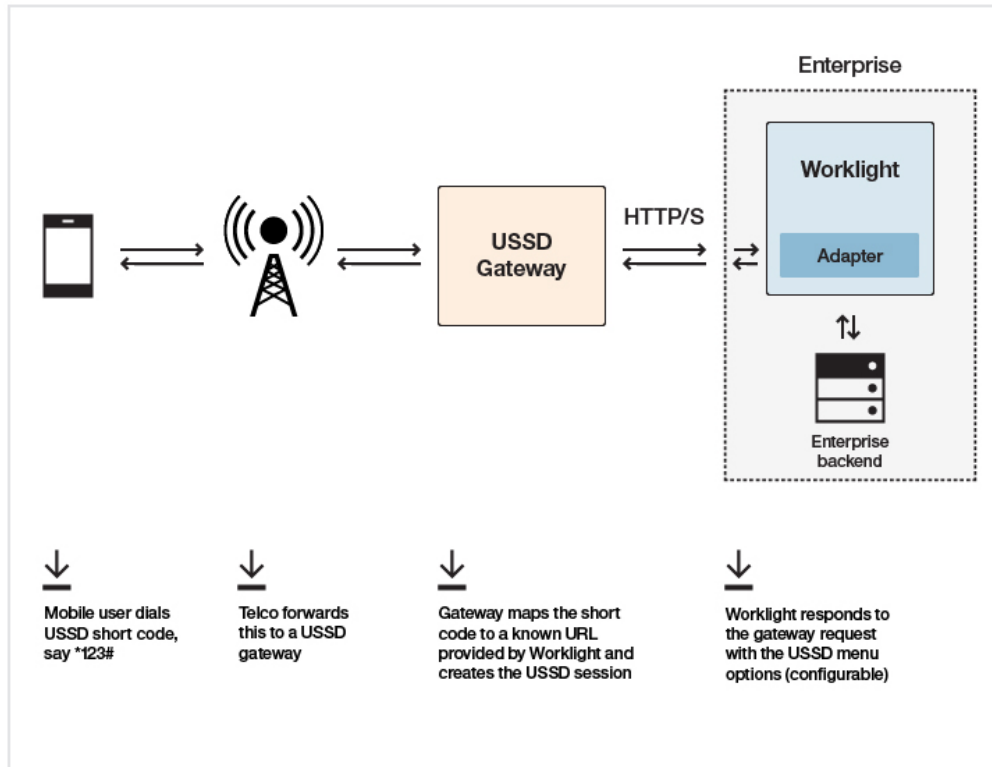
Configuration required at USSD Gateway

```
http://<hostname>:<port>/<contextroot>/ussd
```

This URL can be followed by parameters specific to the gateway. Refer to your USSD Gateway documentation for more details.

Server-side APIs required at Worklight adapter side

To create a filter to process the USSD request:

```
WL.Server.setEventHandlers([ WL.Server.createUSSDEventHandler({
 'shortcode' : '*123#'
}, handleUSSDRequest) ]);
```

To send back a response:

```
WL.Server.createUSSDResponse("This is my response", "text/plain", true))
```

Security

To prevent entities with malicious intent from sending requests to the Worklight Server via a USSD URL, the USSD feature is protected by default. The `authenticationConfig.xml` file is configured to reject all requests to the USSD servlet with a rejecting login module. To allow restricted access to USSD, Worklight administrators must modify the `authenticationConfig.xml` file with appropriate authenticator and login modules, or comment the URL pattern `/ussd*` to allow unrestricted access. For example, the following configuration in the `authenticationConfig.xml` file ensures that only requests with a specific user name in the header of the HTTP request are allowed:

```
<staticResources>
    <resource id="subscribeServlet" securityTest="SubscribeServlet">
        <urlPatterns>/subscribeSMS*;/ussd*</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>
```

Invoking a back-end service for USSD

You can invoke a Worklight HTTP adapter to test the USSD functionality.

Before you begin

This feature is only available within Worklight Studio for HTTP adapters. It is not available when you run an adapter on a stand-alone server that is based on WebSphere Application Server or Tomcat.

About this task

In Worklight Studio, you can invoke an HTTP-based USSD adapter and see the results that are returned to the USSD gateway to verify that the adapter is performing correctly.

Procedure

1. Right-click an adapter file, and select **Run As** > **Invoke Worklight Back-end Service**.
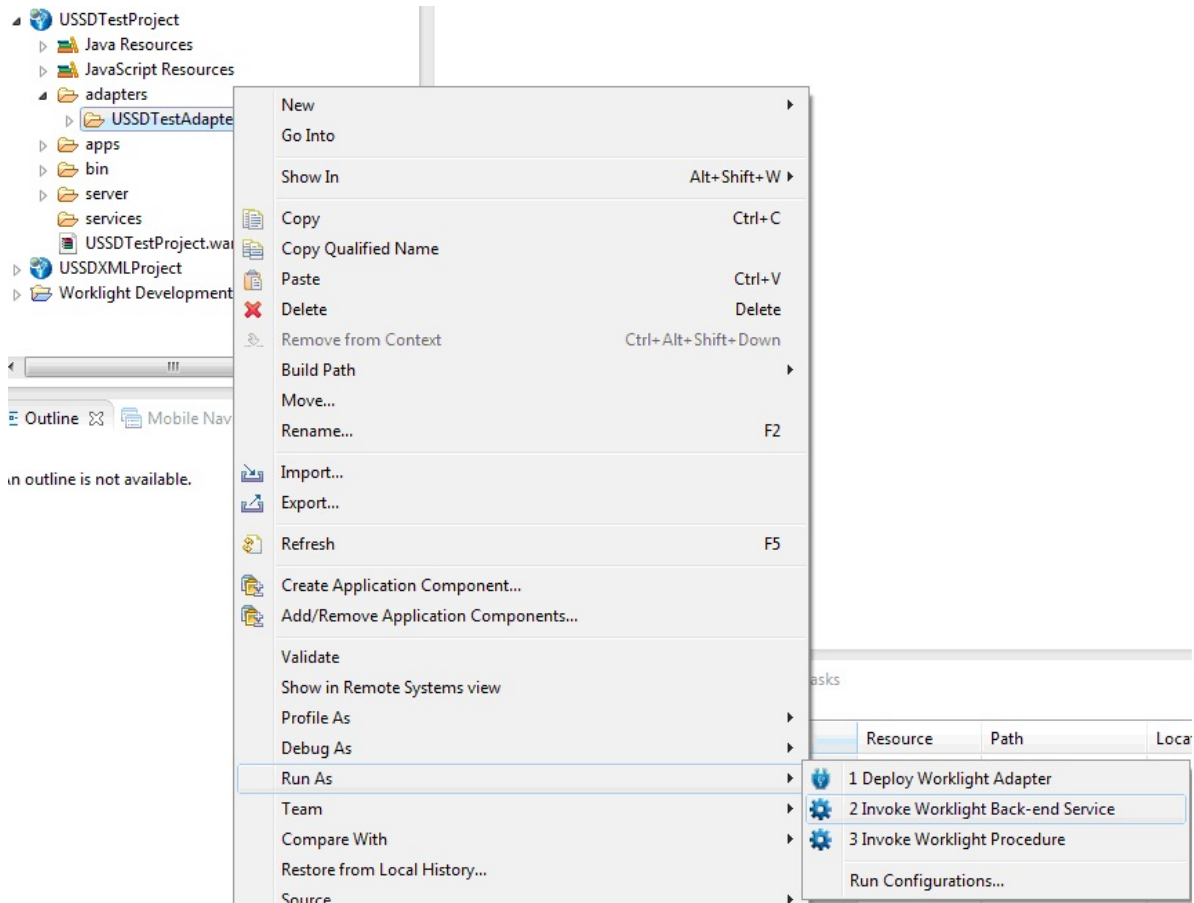
Figure 100. Invoking a Worklight back-end service

2. In the dialog box, from the **Connect** as drop-down list, select **gateway**. Then provide the options for invoking the USSD handler in the text box. The USSD gateway can send HTTP parameters, headers, cookies, or body.
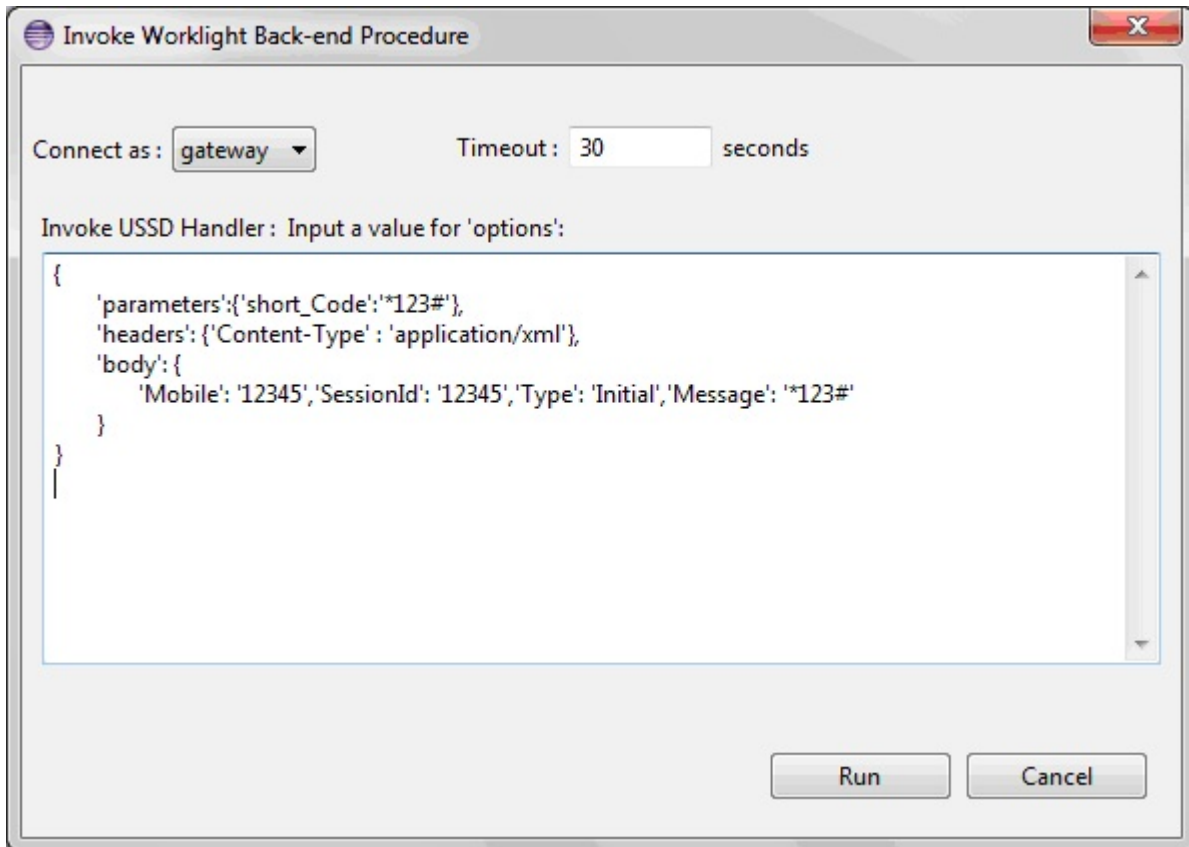
Figure 101. Invocation parameters.

A browser window opens, displaying the result of the adapter invocation

3. You can follow this procedure as many times as required to test the menu flow with the USSD gateway. Here are some examples that use the different types of parameters that are passed from the USSD gateway.

Example 1: Passing query string parameters.
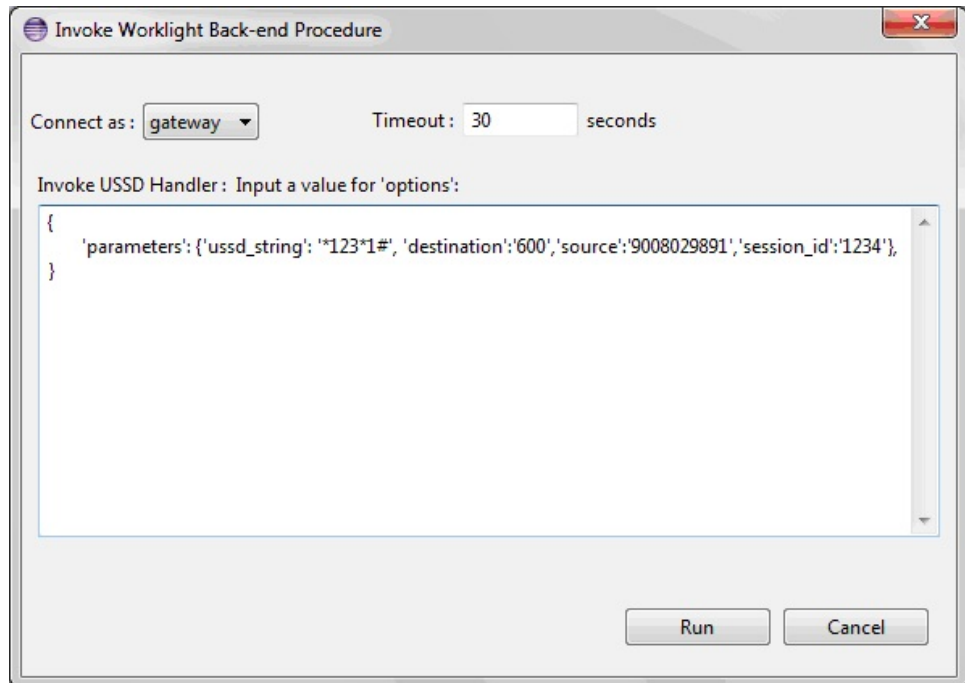
Example 2: Passing JSON parameters in the body.
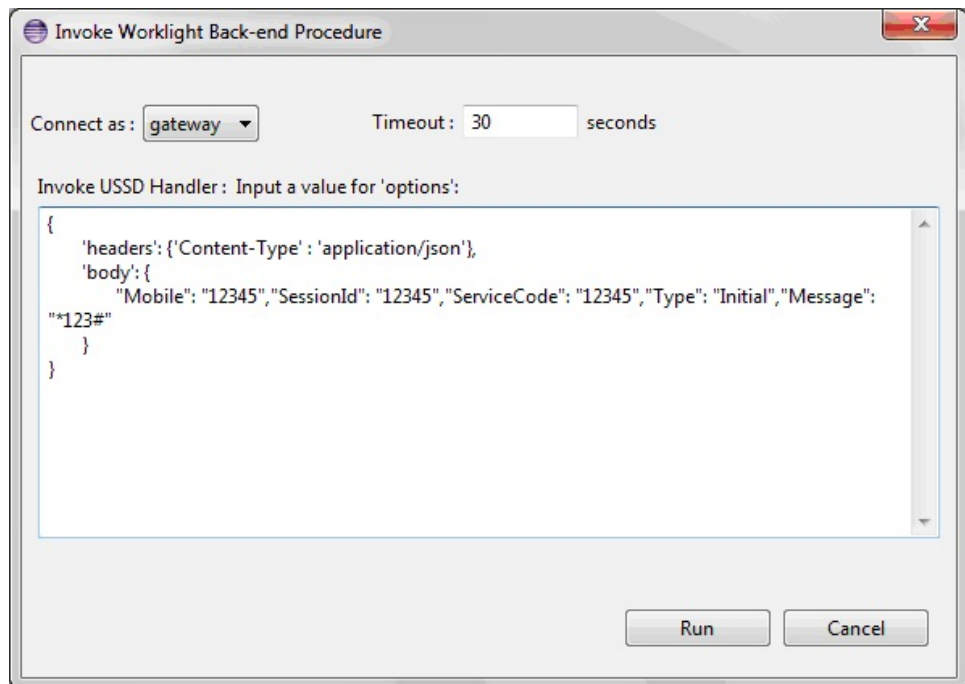
Example 3: Passing XML in the body.
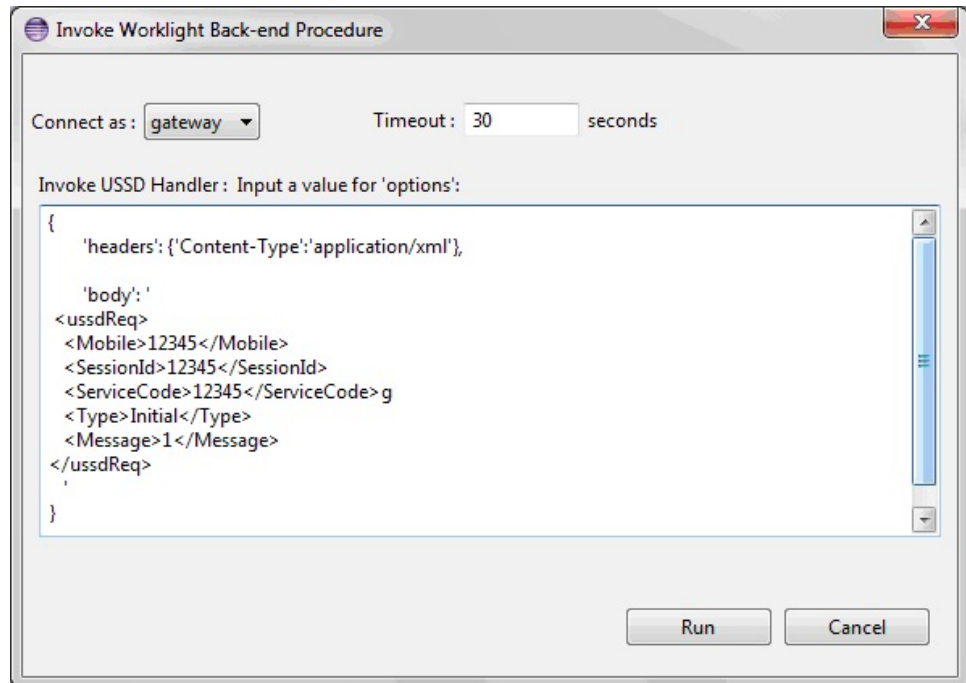
Note: If the body that you pass is not a JSON object, then enclose the object in quotes (" "). If it is a JSON object, then surround it with curly brackets ({ }).

JSONStore

Learn about JSONStore.

JSONStore overview

JSONStore features add the ability to store JSON documents in Worklight applications.

JSONStore is a lightweight, document-oriented storage system that is included as a feature of IBM Worklight Foundation, and enables persistent storage of JSON documents. Documents in an application are available in JSONStore even when the device that is 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 for hybrid applications, see `WL.JSONStore` in the API reference section. Hybrid applications are supported for iOS, Android, Windows 8, and Windows 8 Phone.

For existing customers who have issues with running JSONStore on Windows 8.1 and above, install the Microsoft Visual Studio 2012 C++ runtime library on the development box and reference it in the application. For Microsoft Visual Studio 2013, install the Microsoft Visual Studio 2013 C++ runtime library and update SQLite for Microsoft Visual Studio 2013.

For JSONStore API reference information for native iOS applications, see the `JSONStore` Class Reference in the API reference section.

For JSONStore API reference information for native Android applications, see the `com.worklight.jsonstore.api` Package 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 in hybrid applications, 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 keep track of local changes.

A single store can have many collections, and each collection can have many documents. It is also possible to have a Worklight Application that contains multiple stores. For information, see "JSONStore multiple user support" on page 826.
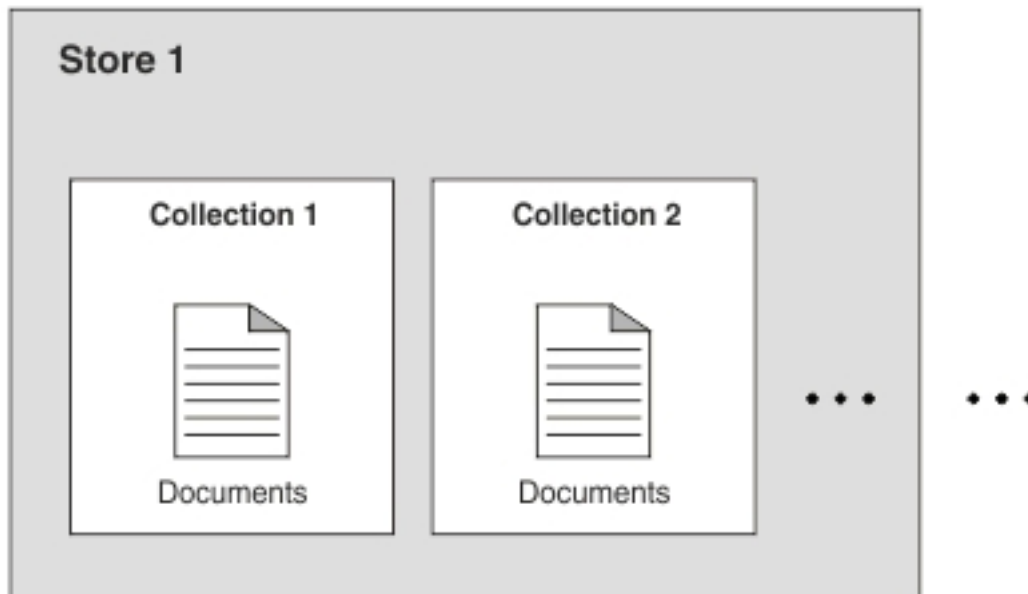
Figure 102. A basic graphic representation of JSONStore.
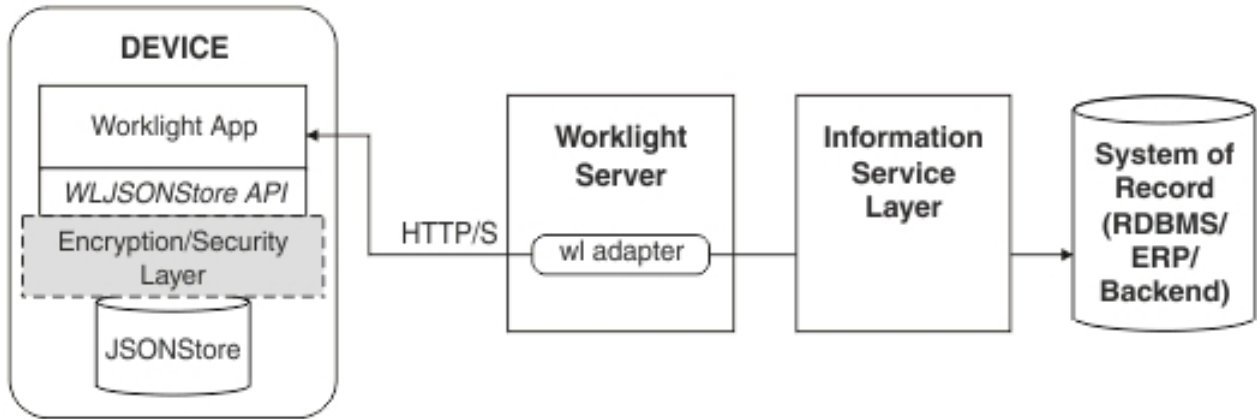
Figure 103. Components and their interaction with the server when you use JSONStore for data synchronization.

Note: Because it is familiar to 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 that is used to store data in relational databases is different from JSONStore's approach. With JSONStore, you can store any JSON content, and index the content that you need to search.

Features table

Compare JSONStore features to those features of other data storage technologies and formats.

JSONStore is a JavaScript API for storing data inside hybrid Worklight applications, an Objective-C API for native iOS applications, and a Java API for native Android applications. For reference, here is a comparison of different JavaScript storage technologies to see how JSONStore compares to them.

JSONStore is similar to technologies such as LocalStorage, Indexed DB, Cordova Storage API, Cordova File API, and Worklight`Encrypted Cache`. The table shows how some features that are provided by JSONStore compare with other technologies. The JSONStore feature is only available on iOS and Android devices and simulators.

Table 140. A comparison of data storage technologies.

	JSONStore	**Encrypted Cache**	**LocalStorage**	**IndexedDB**	**Cordova Storage**	**Cordova File**
Android Support (Hyrbid & Native Applications)	△	△	△	△	△	△
iOS Support (Hybrid & Native Applications)	△	△	△	△	△	△

Table 140. A comparison of data storage technologies. (continued)

	JSONStore	Encrypted Cache	LocalStorage	IndexedDB	Cordova Storage	Cordova File
Windows 8 and Windows 8 Phone 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)	△	-	-	-	△	△
Keep Track of Local Changes	△	-	-	-	-	-
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.

General JSONStore terminology

Learn about general JSONStore terminology.

JSONStore document

A document is the basic building block of JSONStore.

A JSONStore document is a JSON object with an automatically generated identifier (_id) and JSON data. It is similar to a record or a 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. These methods are useful to perform operations 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 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 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. A store 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.

Extra 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 the JSON collection) are indexed and can be used to search more quickly.

Valid data types are: string, boolean, number, and integer. These types 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 extra 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 extra search fields to look
for documents.

The example presumes that the name search field is of type string and the age
search field is of type integer.

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};
```

JSONStore query parts

Query parts are used to build more advanced searches. Some JSONStore
operations, such as some versions of find or count take query parts. Everything
within a query part is joined by AND statements, while query parts themselves are
joined by OR statements. The search criteria returns a match only if everything
within a query part is true. You can use more than one query part to search for
matches that satisfy one or more of the query parts.

Find with query parts operate only on top-level search fields. For example: name,
and not name.first. Use multiple collections where all search fields are top-level to
get around this. The query parts operations that work with non top-level search
fields are: equal, notEqual, like, notLike, rightLike, notRightLike, leftLike, and
notLeftLike. The behavior is undetermined if you use non-top-level search fields.

Enabling JSONStore

To use JSONStore in IBM Worklight V6.0 and later, you must take steps to enable
it.

About this task

To use JSONStore in a native Worklight iOS application, you can import the
`JSONStore.h` header file to use the JSONStore API. For more information about
creating native Worklight iOS applications, see "Developing native applications for
iOS" on page 616.

For native Android applications, you must copy the JAR files that are generated by
IBM Worklight Foundation into your application's `libs` folder. After you do so,
you can use the classes inside the `com.worklight.jsonstore.api` package to use
JSONStore. For more information about creating native Worklight Android
applications, see "Copying files of Native API applications for Android" on page
624.

For hybrid applications, JSONStore is an optional feature in IBM Worklight V6.0
and later. To use JSONStore in a hybrid application, 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 that is titled with your
   application name.
9. Select **Run As...**.
10. Click **Run on Worklight Development Server**.
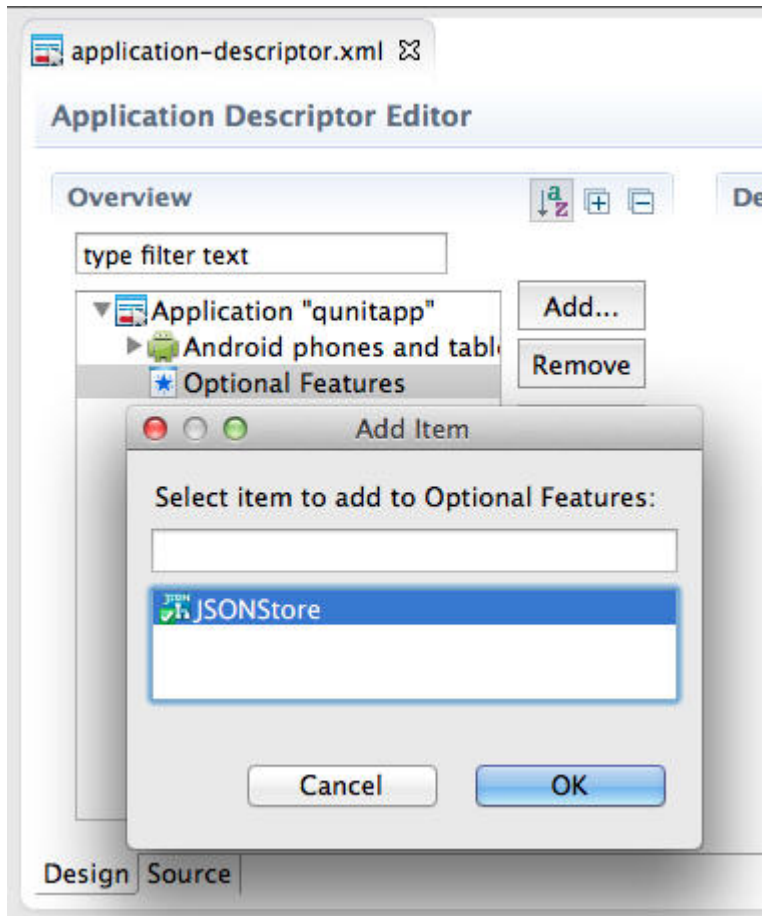
Figure 104. Enabling JSONStore for hybrid applications.

JSONStore API concepts

JSONStore provides API reference information for hybrid Android, iOS, Windows 8, and Windows Phone 8, and native Android and iOS applications.

Store

Open and initialize a collection

Starts one or more collections. Starting or provisioning a JSONStore collection means that the persistent storage that is used to contain collections and documents is created, if it does not exist. If the store is encrypted and a correct password is passed, the required security procedures to make the data accessible are run. There is minimal effort in initializing all the collections when an application starts.

After you open a collection, an accessor to the collection is available, which gives access to collection APIs. It allows developers to call functions such as `find`, `add`, and `replace` on an initialized collection.

It is possible to initialize multiple times with different collections. New collections are initialized without affecting collections that are already initialized.

Destroy

Completely wipes data for all users, destroys the internal storage, and clears security artifacts. The destroy function removes the following data:

- All documents.
- All collections.
- All stores. For more information, see "JSONStore multiple user support" on page 826.
- All JSONStore metadata and security artifacts. For more information, see "JSONStore security" on page 824.

Close all

Locks access to all the collections in a store until the collections are reinitialized. Where initialize can be considered a login, close can be considered a logout.

Start, commit, and rollback transaction

A transaction is a set of operations that must all succeed for the operations to manipulate the store. If any operation fails, the transaction can be rolled back to revert the store to its previous state. After a transaction is started, it is important that you handle committing or rolling back your transactions to prevent excess processing. Three operations exist in the Store API for transactions:

-
  Start transaction

  Begin a snapshot in which the store is reverted to if the transaction fails.

-
  Commit transaction

  Inform the store that all operations in the transaction succeeded, and all changes can be finalized.

-
  Rollback transaction

  Inform the store that an operation in the transaction failed, and all changes must be discarded.

Note: Due to system limitations with multi-threaded transactions, transactions are not supported in Android 2.3.x for hybrid applications. To use transactions in a hybrid application in Android 2.3.x, you can create a Cordova plug-in that uses the native Android JSONStore API to execute the code for the transaction. The whole transaction must be done in the same thread because multi-threaded transactions do not work properly in Android 2.3.x.

Collection

Store and add a document

You can add a document or array of documents to a collection. You can also pass an array of objects (for example [{name: 'carlos'}, {name: 'tim'}]) instead of a single object. Every object in the array is stored as a new document inside the collection.

Remove a document

Marks one or more documents as removed from a collection. Removed documents are not returned by the `find` or `count` operations.

Find All Documents, Find Documents by Id, and Find With Query

You can find documents in a collection by their search fields and extra search fields. An internal search field, _id, holds a unique integer identifier that can be used to find the document (Find by Id). You can search for documents with the following APIs:

*

 Find All Documents
> Returns every document in a collection.

*

 Find All Dirty Documents
> Returns every document in a collection that is marked dirty.

*

 Find by Id
> Find the document with the corresponding _id search key value.

*

 Find With Query or Query Parts
> Find all documents that match a query or all query parts. For more information, see the Search Query format section at "Additional references" on page 797.

Filter returns what is being indexed, which might be different than what was saved to a collection. Some examples of unexpected results are:

1. If your search field has upper case letters, the result is returned in all lower-case letters.
2. If you pass something that is not a string, it is indexed as a string. For example, 1 is '1', 1.0 is '1.0', `true` is '1', and `false` is '0'.
3. If your filter criteria includes non top-level search fields, you might get a single string with all the terms that are joined by a special identifier (-@-). For example, 'carlos-@-mike-@-dgonz'.

Replace a document and change documents

You can use the `Replace` API to replace the contents of a document in the collection with new data, which is based on the _id. If the data contains the _id field of a document in the database, the document is replaced with the data and all search fields are reindexed for that document.

The `Change` API is similar to the `Replace` API, but the `Replace` is based on a set of search field criteria instead of _id. The `Replace` API can be emulated by performing the `Change` API with the search field criteria of only _id. All search fields in the search field criteria must exist in the documents in the store, and in the data that is passed to the `Change` API.

Count All Documents, Count All Dirty Documents, and Count With Query

The `Count` API returns an integer number by counting the total number of documents that match the query. There are three `Count` APIs:

- **Count All Documents**

    Give the total count of all documents in the collection.

- **Count All Dirty Documents**

    Give the total number of documents in the collection that are currently marked dirty.

- **Count With Query or Query Parts**

    Give the total number of documents that match a specific search query. For more information, see the Search Query format section at "Additional references."

Remove Collection and Clear Collection

Removing a collection deletes all data that is associated with a collection, and causes the collection accessor to be no longer usable.

Clearing a collection deletes all documents in the collection. This operation keeps the collection open after it completes.

Mark Clean

The `Mark Clean` API is used to remove the dirty flag from a document in the collection, and deletes the document completely from the collection if it was marked dirty by a remove document operation. The `Mark Clean` API is useful when used with the `Find All Dirty Documents` API to sync the collection with a remote database.

Additional references

Search Query format

When an API requires a search query, a common format is followed for the collection. A query consists of an array of objects where each key/value pair is ANDed together. Each object in the array is ORed together. For example:

```
[{fn: "Mike", age: 30}, {fn: "Carlos", age: 36}]
```

is represented as (with fuzzy search):

```
(fn LIKE "%Mike%" AND age LIKE "%30%") OR (fn LIKE "%Carlos%" AND age LIKE "%36%")
```

Search Query Parts format

The following examples use pseudocode to convey how query parts work. A query such as {name: 'carlos', age: 10} can be passed a modifier such as {exact: true}, which ensures only items that exactly match name and age are returned. Query parts give you the flexibility of adding modifiers to any part of the query. For example:

```
queryPart1 = QueryPart().like('name', 'carlos').lessThan('age', 10);
```

The previous example is transformed into something like:

```
('name' LIKE %carlos%)  AND (age < 10)
```

You can also create another query part, for example:

```
queryPart2 = QueryPart().equal('name', 'mike')
```

When you add various query parts with the `find` API, for example:

```
find([queryPart1, queryPart2]
```

You get something like:

```
( ('name' LIKE %carlos%) AND (age < 10) ) OR (name EQUAL 'mike')
```

Limit and Offset

Passing a limit to an API's options restricts the number of results by the number specified. It is also possible to pass an offset to skip results by the number specified. To pass an offset, a limit must also be passed. This API is useful for implementing pagination or for optimization. By limiting the data to a subset that is necessary, the memory and processing power is reduced.

Fuzzy Search versus Exact Search

The default behavior is fuzzy searching, which means that queries return partial results. For example, the query {name: 'carl'} finds 'carlos' and 'carl' (for example, name LIKE '%carl%'). When {exact: true} is passed, matches are exact but not case-sensitive. For example, 'hello' matches 'Hello' (for example, name.toLowerCase() = 'hello'). Integer matching is not type-sensitive. For example, "1" matches both "1" and "1.0". Numbers are stored as their decimal representation. For example, "1" is stored as "1.0". Boolean values are indexed as 1 (true) and 0 (false).

JSONStore troubleshooting

Find information to help resolve issues that you might encounter when you use the JSONStore API.

JSONStore troubleshooting overview

Find information to help resolve issues that you might encounter when you use the JSONStore API.

Provide information when you ask for help

It is better to provide more information than to risk not providing enough information. The following 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 or Mac OSX 10.8.3.
- Eclipse version. For example, Eclipse Indigo 3.7 Java EE.
- JDK version. For example, Java SE Runtime Environment (build 1.7).
- IBM Worklight Foundation version. For example, IBM Worklight V5.0.6 Developer Edition.
- iOS version. For example, iOS Simulator 6.1 or iPhone 4S iOS 6.0.

- Android version. For example, Android Emulator 4.1.1 or Samsung Galaxy Android 4.0 API Level 14.
- Windows version. For example, Windows 8 or Windows Phone 8.1.
- adb version. For example, Android Debug Bridge version 1.0.31.
- Logs, such as Xcode output on iOS or logcat output on Android.

Try to isolate the issue

Follow these steps to isolate the issue to more accurately report a problem.

1. Reset the emulator (Android) or simulator (iOS) and call the `destroy` API to start with a clean system.
2. Ensure that you are running on a supported production environment.
   - Android >= 2.3 ARM v7/ARM v8/x86 emulator or device
   - iOS >= 6.0 simulator or device
   - Windows Phone 8.0 ARM/x86 emulator or device
   - Windows 8.0-8.1 ARM/x86/x64 simulator or device
3. Try to turn off encryption by not passing a password to the `init` or `open` APIs.
4. Look at the SQLite database file that is generated by JSONStore. Encryption must be turned off.
   - Android emulator:
     ```
     $ adb shell
     $ cd /data/data/com.<app-name>/databases/wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   - iOS simulator:
     ```
     $ cd ~/Library/Application Support/iPhone Simulator/7.1/Applications/<id>/Documents/wljsonst
     $ sqlite3 jsonstore.sqlite
     ```
   - Windows Phone 8:
     ```
     $ cd C:\Data\Users\DefApps\AppData\<id>\Local\wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   - Windows 8 simulator
     ```
     $ cd C:\Users\<username>\AppData\Local\Packages\<id>\LocalState\wljsonstore
     $ sqlite3 jsonstore.sqlite
     ```
   -

     Note: JavaScript only implementation that runs on a web browser (Firefox, Chrome, Safari, Internet Explorer) does not use an SQLite database. The file is stores in HTML5 LocalStorage.
   - Look at the `searchFields` with `.schema` and select data with `SELECT * FROM <collection-name>;`. To exit sqlite3, type `.exit`. If you pass a user name to the `init` method, the file is called *<username>*`.sqlite`. If you do not pass a user name, the file is called `jsonstore.sqlite` by default.
5. (Android only) Enable verbose JSONStore.
   ```
   adb shell setprop log.tag.jsonstore-core VERBOSE
   adb shell getprop log.tag.jsonstore-core
   ```
6. Use the debugger.

Common issues

Understanding the following JSONStore characteristics can help resolve 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`. Store file names or paths instead of the actual files in JSONStore.
- Accessing JSONStore data from native code is possible only in IBM Worklight Foundation V6.2.0.
- There is no limit on how much data you can store inside JSONStore, beyond limits that are imposed by the mobile operating system.
- JSONStore provides persistent data storage. It is not only stored in memory.
- The `init` API fails when the collection name starts with a digit or symbol. IBM Worklight V5.0.6.1 and later returns an appropriate error:

  `4 BAD_PARAMETER_EXPECTED_ALPHANUMERIC_STRING`
- There is a difference between a number and an 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 application is forced to stop or crashes, it always fails with error code `-1` when the application is started again and the `init` or `open` API is called. If this problem happens, call the `closeAll` API first.
- The JavaScript implementation of JSONStore expects code to be called serially. Wait for an operation to finish before you call the next one.
- Transactions are not supported in Android 2.3.x for hybrid applications. For more information, see "JSONStore API concepts" on page 794.
- When you use JSONStore on a 64-bit device, you might see the following error:

  `java.lang.UnsatisfiedLinkError: dlopen failed: "..." is 32-bit instead of 64-bit`

  This error means that you have 64-bit native libraries in your Android project, and JSONStore does not currently work when you use these libraries. To confirm, go to `src/main/libs` or `src/main/jniLibs` under your Android project, and check whether you have the `x86_64` or `arm64-v8a` folders. If you do, delete these folders, and JSONStore can work again.

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 that is stored.
- `name` and `age` are search fields.
- `key` is an extra search field.

Example

Table 141. 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 by 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} }`.

The other internal JSONStore fields are:

_dirty

Determines whether the document was marked as dirty or not. This field is useful to track changes to the documents. For more information, see "JSONStore API concepts" on page 794 or "Work with external data" on page 829.

_deleted

Marks a document as deleted or not. This field is useful to remove objects from the collection, to later use them to track changes with your backend and decide whether to remove them or not.

_operation

A string that reflects the last operation to be performed on the document (for example, replace).

JSONStore errors

Learn about JSONStore errors.

Possible JSONStore error codes that are returned are listed in "JSONStore error codes" on page 802.

JavaScript

JSONStore uses an error object to return messages about the cause of failures.

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.

Example

```
var errorObject = {
  src: 'find', // Operation that failed.
  err: -50, // Error code.
  msg: 'PERSISTENT_STORE_FAILURE', // Error message.
  col: 'people', // Collection name.
  usr: 'jsonstore', // User name.
  doc: {_id: 1, {name: 'carlos', age: 99}}, // Document that is related to the failure.
  res: {...} // Response from the server.
}
```

Not all the key/value pairs are part of every error object. For example, the doc value is only available when the operation failed because of a document (for example the `remove` method in the JSONStoreInstance class) failed to remove a document.

Objective-C

All of the APIs that might fail take an error parameter that takes an address to an NSError object. If you don not want to be notified of errors, you can pass in `nil`. When an operation fails, the address is populated with an NSError, which has an error and some potential `userInfo`. The `userInfo` might contain extra details (for example, the document that caused the failure).

Example

```
// This NSError points to an error if one occurs.
NSError* error = nil;

// Perform the destroy.
[JSONStore destroyDataAndReturnError:&error];
```

Java

All of the Java API calls throw a certain exception, depending on the error that happened. You can either handle each exception separately, or you can catch JSONStoreException as an umbrella for all JSONStore exceptions.

Example

```
try {
  WL.JSONStore.closeAll();
}

catch(JSONStoreException e) {
  // Handle error condition.
}
```

JSONStore error codes
Definitions of the error codes that are related to JSONStore.

-100 UNKNOWN_FAILURE
> Unrecognized error.

-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.

-48 TRANSACTION_FAILURE_DURING_ROLLBACK
> There was a problem with rolling back the transaction.

-47 TRANSACTION_FAILURE_DURING_REMOVE_COLLECTION
> Cannot call removeCollection while a transaction is in progress.

-46 TRANSACTION_FAILURE_DURING_DESTROY
> Cannot call destroy while there are transactions in progress.

-45 TRANSACTION_FAILURE_DURING_CLOSE_ALL
> Cannot call closeAll while there are transactions in place.

-44 TRANSACTION_FAILURE_DURING_INIT
> Cannot initialize a store while there are transactions in progress.

-43 TRANSACTION_FAILURE
> There was a problem with transactions.

-42 NO_TRANSACTION_IN_PROGRESS
> Cannot commit to rolled back a transaction when there is no transaction is progree.

-41 TRANSACTION_IN_POGRESS
> Cannot start a new transaction while another transaction is in progress.

-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 4, "Tutorials and samples," on page 35.

-24 JSON_STORE_FILE_INFO_ERROR
Problem getting the file information from the file system.

-23 JSON_STORE_REPLACE_DOCUMENTS_FAILURE
Problem replacing documents from a collection.

-22 JSON_STORE_REMOVE_WITH_QUERIES_FAILURE
Problem removing documents from a collection.

-21 JSON_STORE_STORE_DATA_PROTECTION_KEY_FAILURE
Problem storing the Data Protection Key (DPK).

-20 JSON_STORE_INVALID_JSON_STRUCTURE
Problem indexing input data.

-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 can
fail while it works 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 first. There can be
only one user at a time.

-5 DESTROY_REMOVE_PERSISTENT_STORE_FAILED
A problem with the destroy method while it tried to delete the file that holds
the contents of the store.

-4 DESTROY_REMOVE_KEYS_FAILED
Problem with the destroy method while it tried 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 before you call
the init or openmethod 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 that tries 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 you call the `add`
method.

20 UNDEFINED_PUSH_OPERATION
No procedure is 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 extra search fields.

22 INVALID_SEARCH_FIELD
One of your search fields is invalid. Verify that none of the search fields that
are passed in 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.

28 INVALID_SORT_OBJECT
The provided array for sorting is invalid because one of the JSON objects is
invalid. The correct syntax is an array of JSON objects, where each object
contains only a single property. This property searches the field with which to
sort, and whether it is ascending or descending. For example: {searchField1 :
"ASC"}.

29 INVALID_FILTER_ARRAY
The provided array for filtering the results is invalid. The correct syntax for
this array is an array of strings, in which each string is either a search field or
an internal JSONStore field. For more information, see "Store internals" on
page 800.

30 BAD_PARAMETER_EXPECTED_ARRAY_OF_OBJECTS
Validation error when the array is not an array of only JSON objects.

31 BAD_PARAMETER_EXPECTED_ARRAY_OF_CLEAN_DOCUMENTS
Validation error.

32 BAD_PARAMETER_WRONG_SEARCH_CRITERIA
Validation error.

JSONStore examples

Learn about how to get started with JSONStore examples.

JavaScript API examples

You can use JSONStore for Worklight hybrid applications.

The following sections contain example implementations for JavaScript with JSONStore APIs. Other helpful topics include:

- IBM Worklight Foundation developerWorks - Questions and answers on Stack Overflow.

Initialize and open connections, get an Accessor, and add data

```
var collectionName = 'people';

// Object that defines all the collections.
var collections = {

  // Object that defines the 'people' collection.
  people : {

    // Object that defines the Search Fields for the 'people' collection.
    searchFields : {name: 'string', age: 'integer'}
  }
};

// Optional options object.
var options = {

  // Optional username, default 'jsonstore'.
  username : 'carlos',

  // Optional password, default no password.
  password : '123',

  // Optional local key generation flag, default false.
  localKeyGen : false
};

WL.JSONStore.init(collections, options)

.then(function () {

  // Data to add, you probably want to get
  // this data from a network call (e.g. Worklight Adapter).
  var data = [{name: 'carlos', age: 10}];

  // Optional options for add.
```

```
  var addOptions = {

    // Mark data as dirty (true = yes, false = no), default true.
    markDirty: true
  };

  // Get an accessor to the people collection and add data.
  return WL.JSONStore.get(collectionName).add(data, addOptions);
})

.then(function (numberOfDocumentsAdded) {
  // Add was successful.
})

.fail(function (errorObject) {
   // Handle failure for any of the previous JSONStore operations (init, add).
});
```

Find - locate documents inside the Store

```
var collectionName = 'people';

// Find all documents that match the queries.
var queryPart1 = WL.JSONStore.QueryPart()
                   .equal('name', 'carlos')
                   .lessOrEqualThan('age', 10)

var options = {
  // Returns a maximum of 10 documents, default no limit.
  limit: 10,

  // Skip 0 documents, default no offset.
  offset: 0,

  // Search fields to return, default: ['_id', 'json'].
  filter: ['_id', 'json'],

  // How to sort the returned values, default no sort.
  sort: [{name: WL.constant.ASCENDING}, {age: WL.constant.DESCENDING}]
};

WL.JSONStore.get(collectionName)

// Alternatives:
// - findById(1, options) which locates documents by their _id field
// - findAll(options) which returns all documents
// - find({'name': 'carlos', age: 10}, options) which finds all documents
// that match the query.
.advancedFind([queryPart1], options)

.then(function (arrayResults) {
  // arrayResults = [{_id: 1, json: {name: 'carlos', age: 99}}]
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Replace - change the documents that are already stored inside a Collection

```
var collectionName = 'people';

// Documents will be located with their '_id' field
// and replaced with the data in the 'json' field.
var docs = [{_id: 1, json: {name: 'carlitos', age: 99}}];
```

```
var options = {

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.replace(docs, options)

.then(function (numberOfDocumentsReplaced) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove - delete all documents that match the query

```
var collectionName = 'people';

// Remove all documents that match the queries.
var queries = [{_id: 1}];

var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true,

  // Mark data as dirty (true = yes, false = no), default true.
  markDirty: true
};

WL.JSONStore.get(collectionName)

.remove(queries, options)

.then(function (numberOfDocumentsRemoved) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Count - gets the total number of documents that match a query

```
var collectionName = 'people';

// Count all documents that match the query.
// The default query is '{}' which will
// count every document in the collection.
var query = {name: 'carlos'};
var options = {

  // Exact match (true) or fuzzy search (false), default fuzzy search.
  exact: true
};

WL.JSONStore.get(collectionName)

.count(query, options)

.then(function (numberOfDocumentsThatMatchedTheQuery) {
  // Handle success.
})
```

```
.fail(function (errorObject) {
  // Handle failure.
});
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
WL.JSONStore.destroy()

.then(function () {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Security - close access to all opened Collections for the current user

```
WL.JSONStore.closeAll()

.then(function () {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
var oldPassword = '123';
var newPassword = '456';

var clearPasswords = function () {
  oldPassword = null;
  newPassword = null;
};

// Default username if none is passed is: 'jsonstore'.
var username = 'carlos';

WL.JSONStore.changePassword(oldPassword, newPassword, username)

.then(function () {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle success.
})

.fail(function (errorObject) {

  // Make sure you do not leave the password(s) in memory.
  clearPasswords();

  // Handle failure.
});
```

Push - get all documents that are marked as dirty, send them to a Worklight adapter, and mark them clean

```
var collectionName = 'people';
var dirtyDocs;

WL.JSONStore.get(collectionName)

.getAllDirty()

.then(function (arrayOfDirtyDocuments) {
  // Handle getAllDirty success.

  dirtyDocs = arrayOfDirtyDocuments;

  var invocationData = {
    adapter : 'adapter-name',
    procedure : 'procedure-name-1',
    parameters : [dirtyDocs],
    compressResponse: true
  };

  return WL.Client.invokeProcedure(invocationData);
})

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // You may want to check the response from the adapter
  // and decide whether or not to mark documents as clean.
  return WL.JSONStore.get(collectionName).markClean(dirtyDocs);
})

.then(function () {
  // Handle markClean success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Pull - get new data from a Worklight adapter

```
var collectionName = 'people';

var invocationData = {
  adapter : 'adapter-name',
  procedure : 'procedure-name-2',
  parameters : [],
  compressResponse: true
};

WL.Client.invokeProcedure(invocationData)

.then(function (responseFromAdapter) {
  // Handle invokeProcedure success.

  // The following example assumes that the adapter returns an arrayOfData,
  // (which is not returned by default),
  // as part of the invocationResult object,
  // with the data that you want to add to the collection.
  var data = responseFromAdapter.invocationResult.arrayOfData;

  // Example:
  // data = [{id: 1, ssn: '111-22-3333', name: 'carlos'}];

  var changeOptions = {
```

```
     // The following example assumes that 'id' and 'ssn' are search fields,
     // default will use all search fields
     // and are part of the data that is received.
     replaceCriteria : ['id', 'ssn'],

     // Data that does not exist in the Collection will be added, default false.
     addNew : true,

     // Mark data as dirty (true = yes, false = no), default false.
     markDirty : false
  };

  return WL.JSONStore.get(collectionName).change(data, changeOptions);
})

.then(function () {
  // Handle change success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check whether a document is dirty

```
var collectionName = 'people';
var doc = {_id: 1, json: {name: 'carlitos', age: 99}};

WL.JSONStore.get(collectionName)

.isDirty(doc)

.then(function (isDocumentDirty) {
  // Handle success.

  // isDocumentDirty - true if dirty, false otherwise.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Check the number of dirty documents

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.countAllDirty()

.then(function (numberOfDirtyDocuments) {
  // Handle success.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Remove a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.removeCollection()

.then(function () {
  // Handle success.
```

```
  // Note: You must call the 'init' API to re-use the empty collection.
  // See the 'clear' API if you just want to remove all data that is inside.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Clear all data that is inside a Collection

```
var collectionName = 'people';

WL.JSONStore.get(collectionName)

.clear()

.then(function () {
  // Handle success.

  // Note: You might want to use the 'removeCollection' API
  // instead if you want to change the search fields.
})

.fail(function (errorObject) {
  // Handle failure.
});
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
WL.JSONStore.startTransaction()

.then(function () {
  // Handle startTransaction success.
  // You can call every JSONStore API method except:
  // init, destroy, removeCollection, and closeAll.

  var data = [{name: 'carlos'}];

  return WL.JSONStore.get(collectionName).add(data);
})

.then(function () {

  var docs = [{_id: 1, json: {name: 'carlos'}}];

  return WL.JSONStore.get(collectionName).remove(docs);
})

.then(function () {

  return WL.JSONStore.commitTransaction();
})

.fail(function (errorObject) {
  // Handle failure for any of the previous JSONStore operation.
  //(startTransaction, add, remove).

  WL.JSONStore.rollbackTransaction()

  .then(function () {
    // Handle rollback success.
  })

  .fail(function () {
```

```
      // Handle rollback failure.
    })

});
```

Get file information

```
WL.JSONStore.fileInfo()
.then(function (res) {
  //res => [{isEncrypted : true, name : carlos, size : 3072}]
})

  .fail(function () {
  // Handle failure.
});
```

Objective-C API examples

You can use JSONStore for Worklight hybrid applications.

The following sections contain example implementations for iOS devices with
JSONStore APIs. Other helpful topics include:

- "JSONStore overview" on page 787 - Learn about key concepts.
- "Enabling JSONStore" on page 792 - Learn how to enable JSONStore in different
  environments.
- "JSONStore API concepts" on page 794 - Learn about general information about
  the APIs that apply to all implementations of the JSONStore API.
- "JSONStore troubleshooting" on page 798 - Learn how to debug and understand
  possible errors.
- "JSONStore advanced topics" on page 824 - Learn about security, multiple user
  support, performance, and concurrency.
- JSONStore Class Reference - Learn about JSONStore APIs for Objective-C.
- "Work with external data" on page 829 - Explains how to get data from an
  external source and send changes back to the external source.
- IBM Worklight Foundation developerWorks - Questions and answers on Stack
  Overflow.

Initialize and open connections, get an Accessor, and add data

```
// Create the collections object that will be initialized.
JSONStoreCollection* people = [[JSONStoreCollection alloc] initWithName:@"people"];
[people setSearchField:@"name" withType:JSONStore_String];
[people setSearchField:@"age" withType:JSONStore_Integer];

// Optional options object.
JSONStoreOpenOptions* options = [JSONStoreOpenOptions new];
[options setUsername:@"carlos"]; //Optional username, default 'jsonstore'
[options setPassword:@"123"]; //Optional password, default no password

// This object will point to an error if one occurs.
NSError* error = nil;

// Open the collections.
[[JSONStore sharedInstance] openCollections:@[people] withOptions:options error:&error];

// Add data to the collection
NSArray* data = @[ @{@"name" : @"carlos", @"age": @10} ];
int newDocsAdded = [[people addData:data andMarkDirty:YES withOptions:nil error:&error] intValue];
```

Initialize with a secure random token from the server

```
[WLSecurityUtils getRandomStringFromServerWithBytes:32
                timeout:1000
                completionHandler:^(NSURLResponse *response,
                                   NSData *data,
                                   NSError *connectionError) {
```

```
        // You might want to see the response and the connection error
        // before moving forward.

        // Get the secure random string by using the data that is
        // returned from the generator on the server.
        NSString* secureRandom = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];

        JSONStoreCollection* ppl = [[JSONStoreCollection alloc] initWithName:@"people"];
        [ppl setSearchField:@"name" withType:JSONStore_String];
        [ppl setSearchField:@"age" withType:JSONStore_Integer];

        // Optional options object.
        JSONStoreOptions* options = [JSONStoreOptions new];
        [options setUsername:@"carlos"]; //Optional username, default 'jsonstore'
        [options setPassword:@"123"]; //Optional password, default no password
        [options setSecureRandom:secureRandom]; //Optional, default one will be generated locally

        // This points to an error if one occurs.
        NSError* error = nil;

        [[JSONStore sharedInstance] openCollections:@[ppl] withOptions:options error:&error];

        // Other JSONStore operations (e.g. add, remove, replace, etc.) go here.
}];
```

Find - locate documents inside the Store

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Add additional find options (optional).
JSONStoreQueryOptions* options = [JSONStoreQueryOptions new];
[options setLimit:@10]; // Returns a maximum of 10 documents, default no limit.
[options setOffset:@0]; // Skip 0 documents, default no offset.

// Search fields to return, default: ['_id', 'json'].
[options filterSearchField:@"_id"];
[options filterSearchField:@"json"];

// How to sort the returned values , default no sort.
[options sortBySearchFieldAscending:@"name"];
[options sortBySearchFieldDescending:@"age"];

// Find all documents that match the query part.
JSONStoreQueryPart* queryPart1 = [[JSONStoreQueryPart alloc] init];
[queryPart1 searchField:@"name" equal:@"carlos"];
[queryPart1 searchField:@"age" lessOrEqualThan:@10];

NSArray* results = [people findWithQueryParts:@[queryPart1] andOptions:options error:&error];

// results = @[ @{@"_id" : @1, @"json" : @{ @"name": @"carlos", @"age" : @10}} ];

for (NSDictionary* result in results) {

  NSString* name = [result valueForKeyPath:@"json.name"]; // carlos.
  int age = [[result valueForKeyPath:@"json.age"] intValue]; // 10
  NSLog(@"Name: %@, Age: %d", name, age);
}
```

Replace - change the documents that are already stored inside a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// Find all documents that match the queries.
NSArray* docs = @[ @{@"_id" : @1, @"json" : @{ @"name": @"carlitos", @"age" : @99}} ];

// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the replacement.
int docsReplaced = [[people replaceDocuments:docs andMarkDirty:NO error:&error] intValue];
```

Remove - delete all documents that match the query

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Find document with _id equal to 1 and remove it.
int docsRemoved = [[people removeWithIds:@[@1] andMarkDirty:NO error:&error] intValue];
```

Count - gets the total number of documents that match a query

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// Count all documents that match the query.
// The default query is @{} which will
// count every document in the collection.
JSONStoreQueryPart *queryPart = [[JSONStoreQueryPart alloc] init];
[queryPart searchField:@"name" equal:@"carlos"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the count.
int countResult = [[people countWithQueryParts:@[queryPart] error:&error] intValue];
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the destroy.
[[JSONStore sharedInstance] destroyDataAndReturnError:&error];
```

Security - close access to all opened Collections for the current user

```
// This object will point to an error if one occurs.
NSError* error = nil;

// Close access to all collections in the store.
[[JSONStore sharedInstance] closeAllCollectionsAndReturnError:&error];
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hardcoded in the example for brevity.
NSString* oldPassword = @"123";
NSString* newPassword = @"456";
NSString* username = @"carlos";

// This object will point to an error if one occurs.
NSError* error = nil;

// Perform the change password operation.
[[JSONStore sharedInstance] changeCurrentPassword:oldPassword withNewPassword:newPassword forUsername:username error:&error];

// Remove the passwords from memory.
oldPassword = nil;
newPassword = nil;
```

Push - get all documents that are marked as dirty, send them to a Worklight adapter, and mark them clean

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs
NSError* error = nil;

// Return all documents marked dirty
NSArray* dirtyDocs = [people allDirtyAndReturnError:&error];

// ACTION REQUIRED: Handle the dirty documents here
// (e.g. send them to a Worklight Adapter).

// Mark dirty documents as clean
int numCleaned = [[people markDocumentsClean:dirtyDocs error:&error] intValue];
```

Pull - get new data from a Worklight adapter

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;


// ACTION REQUIRED: Get data (e.g. Worklight Adapter).
// For this example, it is hardcoded.
NSArray* data = @[ @{@"id" : @1, @"ssn": @"111-22-3333", @"name": @"carlos"} ];


int numChanged = [[people changeData:data withReplaceCriteria:@[@"id", @"ssn"] addNew:YES markDirty:NO error:&error] intValue];
```

Check whether a document is dirty

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Check if document with _id '1' is dirty.
BOOL isDirtyResult = [people isDirtyWithDocumentId:1 error:&error];
```

Check the number of dirty documents

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Check if document with _id '1' is dirty.
int dirtyDocsCount = [[people countAllDirtyDocumentsWithError:&error] intValue];
```

Remove a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Remove the collection.
[people removeCollectionWithError:&error];
```

Clear all data that is inside a Collection

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// This object will point to an error if one occurs.
NSError* error = nil;

// Remove the collection.
[people clearCollectionWithError:&error];
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
// Get the accessor to an already initialized collection.
JSONStoreCollection* people = [[JSONStore sharedInstance] getCollectionWithName:@"people"];

// These objects will point to errors if they occur.
NSError* error = nil;
NSError* addError = nil;
NSError* removeError = nil;

// You can call every JSONStore API method inside a transaction except:
// open, destroy, removeCollection and closeAll.
[[JSONStore sharedInstance] startTransactionAndReturnError:&error];

[people addData:@[ @{@"name" : @"carlos"} ] andMarkDirty:NO withOptions:nil error:&addError];

[people removeWithIds:@[@1] andMarkDirty:NO error:&removeError];

if (addError != nil || removeError != nil) {

  // Return the store to the state before start transaction was called.
  [[JSONStore sharedInstance] rollbackTransactionAndReturnError:&error];
```

```
} else {
  // Commit the transaction thus ensuring atomicity.
  [[JSONStore sharedInstance] commitTransactionAndReturnError:&error];
}
```

Get file information

```
// This object will point to an error if one occurs
NSError* error = nil;

// Returns information about files JSONStore uses to persist data.
NSArray* results = [[JSONStore sharedInstance] fileInfoAndReturnError:&error];
// => [{@"isEncrypted" : @(true), @"name" : @"carlos", @"size" : @3072}]
```

Java API examples

You can use JSONStore for Worklight hybrid applications.

The following sections contain example implementations for Android devices with JSONStore APIs. Other helpful topics include:

Initialize and open connections, get an Accessor, and add data

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
  // Create the collections object that will be initialized.
  JSONStoreCollection peopleCollection = new JSONStoreCollection("people");
  peopleCollection.setSearchField("name", SearchFieldType.STRING);
  peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
  collections.add(peopleCollection);

  // Optional options object.
  JSONStoreInitOptions initOptions = new JSONStoreInitOptions();
  // Optional username, default 'jsonstore'.
  initOptions.setUsername("carlos");
  // Optional password, default no password.
  initOptions.setPassword("123");

  // Open the collection.

  WLJSONStore.getInstance(ctx).openCollections(collections, initOptions);

  // Add data to the collection.
  JSONObject newDocument = new JSONObject("{name: 'carlos', age: 10}");
  JSONStoreAddOptions addOptions = new JSONStoreAddOptions();
  addOptions.setMarkDirty(true);
```

```
      peopleCollection.addData(newDocument, addOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations (init, add).
  throw ex;
} catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
throw ex;
}
```

Initialize with a secure random token from the server

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

// Do an AsyncTask because networking cannot occur inside the activity.
AsyncTask<Context, Void, Void> aTask = new AsyncTask<Context, Void, Void>() {
  protected Void doInBackground(Context... params) {
    final Context context = params[0];

    // Create the request listener that will have the
    // onSuccess and onFailure callbacks:
    WLRequestListener listener = new WLRequestListener() {
      public void onFailure(WLFailResponse failureResponse) {
        // Handle Failure.
      }

      public void onSuccess(WLResponse response) {
        String secureRandom = response.getResponseText();

        try {
          List<JSONStoreCollection> collections = new LinkedList<JSONStoreCollection>();
          // Create the collections object that will be initialized.
          JSONStoreCollection peopleCollection = new JSONStoreCollection("people");
          peopleCollection.setSearchField("name", SearchFieldType.STRING);
          peopleCollection.setSearchField("age", SearchFieldType.INTEGER);
          collections.add(peopleCollection);

          // Optional options object.
          JSONStoreInitOptions initOptions = new JSONStoreInitOptions();

          // Optional username, default 'jsonstore'.
          initOptions.setUsername("carlos");

          // Optional password, default no password.
          initOptions.setPassword("123");

          initOptions.setSecureRandom(secureRandom);

          // Open the collection.
          WLJSONStore.getInstance(context).openCollections(collections, initOptions);

          // Other JSONStore operations (e.g. add, remove, replace, etc.) go here.
        }
        catch (JSONStoreException ex) {
          // Handle failure for any of the previous JSONStore operations (init, add).
          ex.printStackTrace();        }
      }
    };

    // Get the secure random from the server:
    // The length of the random string, in bytes (maximum is 64 bytes).
    int byteLength = 32;
    SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
```

```
      return null;
    }
  };
};
aTask.execute(ctx);
```

Find - locate documents inside the Store

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people

  JSONStoreQueryParts findQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();
  part.addLike("name", "carlos");
  part.addLessThan("age", 99);
  findQuery.addQueryPart(part);

  // Add additional find options (optional).
  JSONStoreFindOptions findOptions = new JSONStoreFindOptions();

  // Returns a maximum of 10 documents, default no limit.
  findOptions.setLimit(10);
  // Skip 0 documents, default no offset.
  findOptions.setOffset(0);

  // Search fields to return, default: ['_id', 'json'].
  findOptions.addSearchFilter("_id");
  findOptions.addSearchFilter("json");

  // How to sort the returned values, default no sort.
  findOptions.sortBySearchFieldAscending("name");
  findOptions.sortBySeachFieldDescending("age");

  // Find documents that match the query.
  List<JSONObject> results = peopleCollection.findDocuments(findQuery, findOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Replace - change the documents that are already stored inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people

  // Documents will be located with their '_id' field
  //and replaced with the data in the 'json' field.
  JSONObject replaceDoc = new JSONObject("{_id: 1, json: {name: 'carlitos', age: 99}}");

  // Mark data as dirty (true = yes, false = no), default true.
  JSONStoreReplaceOptions replaceOptions = new JSONStoreReplaceOptions();
  replaceOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.replaceDocument(replaceDoc, replaceOptions);
}
```

```
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove - delete all documents that match the query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Documents will be located with their '_id' field.
  int id = 1;

  JSONStoreRemoveOptions removeOptions = new JSONStoreRemoveOptions();

  // Mark data as dirty (true = yes, false = no), default true.
  removeOptions.setMarkDirty(true);

  // Replace the document.
  peopleCollection.removeDocumentById(id, removeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Count - gets the total number of documents that match a query

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Count all documents that match the query.
  JSONStoreQueryParts countQuery = new JSONStoreQueryParts();
  JSONStoreQueryPart part = new JSONStoreQueryPart();

  // Exact match.
  part.addEqual("name", "carlos");
  countQuery.addQueryPart(part);

  // Replace the document.
  int resultCount = peopleCollection.countDocuments(countQuery);
  JSONObject doc = peopleCollection.findDocumentById(resultCount);
  peopleCollection.replaceDocument(doc);
}
catch (JSONStoreException ex) {
  throw ex;
}
```

Destroy - wipes data for all users, destroys the internal storage, and clears security artifacts

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Destroy the Store.
```

```
    WLJSONStore.getInstance(ctx).destroy();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Security - close access to all opened Collections for the current user

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Close access to all collections.
  WLJSONStore.getInstance(ctx).closeAll();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Security - change the password that is used to access a Store

```
// The password should be user input.
// It is hard-coded in the example for brevity.
String username = "carlos";
String oldPassword = "123";
String newPassword = "456";

// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  WLJSONStore.getInstance(ctx).changePassword(oldPassword, newPassword, username);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
finally {
  // It is good practice to not leave passwords in memory
  oldPassword = null;
  newPassword = null;
}
```

Push - get all documents that are marked as dirty, send them to a Worklight adapter, and mark them clean

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection  = WLJSONStore.getInstance(ctx).getCollectionByName("people

  // Check if document with _id 3 is dirty.
  List<JSONObject> allDirtyDocuments = peopleCollection.findAllDirtyDocuments();

  // Handle the dirty documents here (e.g. calling an adapter).

  peopleCollection.markDocumentsClean(allDirtyDocuments);
}  catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations
  throw ex;
}
```

Pull - get new data from a Worklight adapter

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Pull data here and place in newDocs. For this example, it is hard-coded.
  List<JSONObject> newDocs = new ArrayList<JSONObject>();
  JSONObject doc = new JSONObject("{id: 1, ssn: '111-22-3333', name: 'carlos'}");
  newDocs.add(doc);

  JSONStoreChangeOptions changeOptions = new JSONStoreChangeOptions();

  // Data that does not exist in the collection will be added, default false.
  changeOptions.setAddNew(true);

  // Mark data as dirty (true = yes, false = no), default false.
  changeOptions.setMarkDirty(true);

  // The following example assumes that 'id' and 'ssn' are search fields,
  // default will use all search fields
  // and are part of the data that is received.
  changeOptions.addSearchFieldToCriteria("id");
  changeOptions.addSearchFieldToCriteria("ssn");

  int changed = peopleCollection.changeData(newDocs, changeOptions);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.
  throw ex;
}
```

Check whether a document is dirty

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Check if document with id '3' is dirty.
  boolean isDirty = peopleCollection.isDocumentDirty(3);
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Check the number of dirty documents

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people")

  // Get the count of all dirty documents in the people collection.
  int totalDirty = peopleCollection.countAllDirtyDocuments();
}
```

```
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Remove a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people

  // Remove the collection. The collection object is
  // no longer usable.
  peopleCollection.removeCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Clear all data that is inside a Collection

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people

  // Clear the collection.
  peopleCollection.clearCollection();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.
  throw ex;
}
```

Start a transaction, add some data, remove a document, commit the transaction and roll back the transaction if there is a failure

```
// Fill in the blank to get the Android application context.
Context ctx = getContext();

try {
  // Get the already initialized collection.
  JSONStoreCollection peopleCollection = WLJSONStore.getInstance(ctx).getCollectionByName("people

  WLJSONStore.getInstance(ctx).startTransaction();

  JSONObject docToAdd = new JSONObject("{name: 'carlos', age: 99}");
  // Find documents that match query.
  peopleCollection.addData(docToAdd);


  //Remove added doc.
  int id = 1;
  peopleCollection.removeDocumentById(id);

  WLJSONStore.getInstance(ctx).commitTransaction();
}
catch (JSONStoreException ex) {
  // Handle failure for any of the previous JSONStore operations.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();
```

```
    throw ex;
}
catch (JSONException ex) {
  // Handle failure for any JSON parsing issues.

  // An exception occured. Take care of it to prevent further damage.
  WLJSONStore.getInstance(ctx).rollbackTransaction();

  throw ex;
}
```

Get file information

```
Context ctx = getContext();
List<JSONStoreFileInfo> allFileInfo = WLJSONStore.getInstance(ctx).getFileInfo();

for(JSONStoreFileInfo fileInfo : allFileInfo) {
  long fileSize = fileInfo.getFileSizeBytes();
  String username = fileInfo.getUsername();
  boolean isEncrypted = fileInfo.isEncrypted();
}
```

JSONStore advanced topics

Learn about JSONStore advanced topics.

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 (JavaScript) or open (Native iOS and Native Android) API. If no password is passed, none of the documents in the store collections are encrypted.

Some security artifacts (for example salt) are stored in the keychain (iOS), shared preferences (Android), isolated storage (Windows 8 Phone), or the credential locker (Windows 8). 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, iOS, Windows 8 Phone, and Windows 8 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 API.

The 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 puts in the application, use the destroy API. This process is not applicable to Android because the encrypted DPK is stored in shared preferences and wiped out when the application is uninstalled.

The first time that JSONStore opens a collection with a password, which means that the developer wants to encrypt data inside the store, JSONStore needs a random token. That random token can be obtained from the client or from the server.

When the `localKeyGen` key is present in the JavaScript implementation of the JSONStore API, and it has a value of `true`, a cryptographically secure token is generated locally. Otherwise, the token is generated by contacting the server, thus requiring connectivity to the Worklight Server. This token is required only the first time that a store is opened with a password. The native implementations (Objective-C and Java) generate a cryptographically secure token locally by default, or you can pass one through the `secureRandom` option.

The trade-off is between opening a store offline and trusting the client to generate that random token (less secure), or opening the store with access to the Worklight Server (requires connectivity) and trusting the server (more secure).

Windows 8 and Windows Phone 8 encryption:

You can secure all of the collections in a store by encrypting them.

JSONStore uses SQLCipher as its underlying database technology. SQLCipher is a build of SQLite that is produced by Zetetic, LLC that adds a layer of encryption to the database.

JSONStore uses SQLCipher on all platforms. On Android and iOS a free, open source version of SQLCipher is available, known as the Community Edition and is incorporated into the versions of JSONStore that is included in IBM Worklight Foundation. The Windows versions of SQLCipher are only available under a commercial license and cannot be directly redistributed by IBM Worklight Foundation.

Instead, JSONStore for Windows 8 and Windows Phone 8 include SQLite as the underlying database. If you need to encrypt data for either of these platforms, you need to acquire your own version of SQLCipher and swap out the SQLite version that is included in IBM Worklight Foundation. For more information, see "SQLCipher on Windows Phone 8 and Windows 8 or 8.1."

If you do not need encryption, the JSONStore is fully functional (minus encryption) by using the SQLite version in IBM Worklight Foundation.

SQLCipher on Windows Phone 8 and Windows 8 or 8.1:

To use JSONStore encryption on Windows, you must replace SQLite with SQLCipher.

Replacing SQLite with SQLCipher for Windows Phone 8:

To use JSONStore encryption on Windows Phone 8, you must replace SQLite with SQLCipher.

Procedure
1. Run the SQLCipher for Windows Phone extension that comes with the SQLCipher for Windows Phone Commercial Edition.
2. After the extension finishes installing, locate the SQLCipher version of the `sqlite3.dll` file that was just created. There is one for x86, one for x64, and one for ARM.
   `C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v8.0\ExtensionSDKs\SQLCipher.WP80\3.0.1\Redist\Retail\<platform>`
3. Copy and replace this file to your Worklight application.
   `<Worklight project name>\apps\<application name>\windowsphone8\native\buildtarget\<platform>`

Replacing SQLite with SQLCipher for Windows 8:

To use JSONStore encryption on Windows 8, you must replace SQLite with SQLCipher.

Procedure

1. Run the SQLCipher for Windows Runtime 8.0 extension that comes with the SQLCipher for Windows Runtime Commercial Edition.
2. After the extension finishes installing, locate the SQLCipher version of the `sqlite3.dll` file that was just created. There is one for x86, one for x64, and one for ARM.

   `C:\Program Files (x86)\Microsoft SDKs\Windows\v8.0\ExtensionSDKs\SQLCipher.WinRT80\3.0.1\Redist\Retail\`*`<platform>`*
3. Copy and replace this file to your Worklight application.

   `<Worklight project name>``\apps\`*`<application name>`*`\windows8\native\buildtarget\`*`<platform>`*

Replacing SQLite with SQLCipher for Windows 8.1:

To use JSONStore encryption on Windows 8.1, you must replace SQLite with SQLCipher.

Procedure

1. Run the SQLCipher for Windows Runtime 8.1 extension that comes with the SQLCipher for Windows Runtime Commercial Edition.
2. After the extension finishes installing, locate the SQLCipher version of the `sqlite3.dll` file that was just created. There is one for x86, one for x64, and one for ARM.

   `C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1\ExtensionSDKs\SQLCipher.WinRT81\3.0.1\Redist\Retail\`*`<platform>`*
3. Copy and replace this file to your Worklight application.

   `<Worklight project name>``\apps\`*`<application name>`*`\windows8\native\buildtarget\`*`<platform>`*

JSONStore multiple user support

With JSONStore, you can create multiple stores that contain different collections in a single Worklight application.

The `init` (JavaScript) or `open` (Native iOS and Native Android) API can take an options object with a user name. Different stores are separate files in the file system. The user name is used as the file name of the store. These separate stores can be encrypted with different passwords for security and privacy reasons. Calling the `closeAll` API removes access to all the collections. It is also possible to change the password of an encrypted store by calling the `changePassword` API.

An example use case would be various employees that share a physical device (for example an iPad or Android tablet) and Worklight application. In addition, if the employees work different shifts and handle private data from different customers while they use the Worklight application, multiple user support is useful.

JSONStore performance

Learn about the factors that can affect JSONStore performance.

Network

- IBM Worklight Foundation provides APIs for getting information about the network, for example, `WL.Device.getNetworkInfo` (JavaScript). Ideally, getting and sending data from and to a Worklight adapter should be done when the application is using a WiFi network.

- Check network connectivity before you perform operations, such as sending all dirty documents to a Worklight adapter.
- The amount of data that is sent over the network to a client heavily affects performance. Send only the data that is required by the application, instead of copying everything inside your backend database.
- If you are using a Worklight adapter, consider setting the `compressResponse` flag to `true`. That way, responses are compressed, which generally uses less bandwidth and has a faster transfer time than without compression.

Memory

- When you use the JavaScript API, JSONStore documents are serialized and deserialized as Strings between the Native (Objective-C, Java, or C#) Layer and the JavaScript Layer. One way to mitigate possible memory issues is by using limit and offset when you use the `find` API. That way, you limit the amount of memory that is allocated for the results and can implement things like pagination (show X number of results per page).
- 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, you map them to short key names when you send them from the adapter to the client, and map them to the original long key names when you send data back to the backend.
- Consider splitting the data inside a store into various collections. Have small documents over various collections instead of monolithic documents in a single collection. This consideration depends on how closely related the data is and the use cases for said data.
- When you use the `add` API with an array of objects, it is possible to run into memory issues. To mitigate this issue, call these methods with fewer JSON objects at a time.
- JavaScript and Java have garbage collectors, while Objective-C has Automatic Reference Counting. 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 expect it to go down.

CPU

- The amount of search fields and extra search fields that are used affect performance when you call the `add` method, which does the indexing. Only index the values that are used in queries for the `find` method.
- By default, JSONStore tracks local changes to its documents. This behavior can be disabled, thus saving a few cycles, by setting the `markDirty` flag to `false` when you use the `add`, `remove`, and `replace` APIs.
- Enabling security adds some overhead to the `init` or `open` APIs and other operations that work with documents inside the collection. Consider whether security is genuinely required. For example, the `init` or `open` APIs are much slower with encryption, as they must generate the encryption keys that they use for encryption and decryption.
- The `replace` and `remove` APIs depend on the collection size as they must go through the whole collection to replace or 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 hit is more noticeable on large collections.
- The `count` API is relatively expensive. However, you can keep a variable that keeps the count for that collection. Update it every time that you store or remove things from the collection.

- The find APIs (find, findAll, and findById) 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.

More information

For more information about JSONStore performance, see the JSONStore performance blog post.

JSONStore concurrency
Learn about JSONStore concurrency.

JavaScript

Most of the operations that can be performed on a collection, such as add and find, are asynchronous. These operations return a jQuery promise that is resolved when the operation completes 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 following examples are not specific to JSONStore, but are intended to help you understand their usage in general.

The Options Object with onSuccess and onFailure callbacks that were used in JSONStore for IBM Worklight V5.0.5 are deprecated in favor of promises.

Instead of promises and callbacks, you can also listen to JSONStore success ('WL/JSONSTORE/SUCCESS') and failure ('WL/JSONSTORE/FAILURE' events. Perform actions that are based on the arguments that are passed to the event listener.

Example promise definition
```
var asyncOperation = function () {
  // Assumes that you have jQuery defined via $ in the environment
  var deferred = $.Deferred();

  setTimeout(function() {
    deferred.resolve('Hello');
  }, 1000);

  return deferred.promise();
};
```

Example promise usage
```
// The function that is passed to .then is executed after 1000 ms.
asyncOperation.then(function (response) {
  // response = 'Hello'
});
```

Example callback definition
```
var asyncOperation = function (callback) {
  setTimeout(function() {
    callback('Hello');
  }, 1000);
};
```

Example callback usage
```
// The function that is passed to asyncOperation is executed after 1000 ms.
asyncOperation(function (response) {
  // response = 'Hello'
});
```

Example events
```
$(document.body).on('WL/JSONSTORE/SUCCESS', function (evt, data, src, collectionName) {

   // evt - Contains information about the event
   // data - Data that is sent ater the operation (add, find, etc.) finished
   // src - Name of the operation (add, find, push, etc.)
   // collectionName - Name of the collection
});
```

Objective-C

When you use the Native iOS API for JSONStore, all operations are added to a synchronous dispatch queue. This behavior ensures that operations that touch the store are executed in order on a thread that is not the main thread. For more information, see the Apple documentation at Grand Central Dispatch (GCD).

Java

When you use the Native Android API for JSONStore, all operations are executed on the main thread. You must create threads or use thread pools to have asynchronous behavior. All store operations are thread-safe.

Work with external data

Learn about the different concepts that are required to work with external data.

For the actual API examples, see "JSONStore examples" on page 806.

Pull

Many systems use the term *pull* to refer to getting data from an external source.

There are three important pieces:

External Data Source
> This source can be a database, a REST or SOAP API, or many others. The only requirement is that it must be accessible from either the Worklight Server or directly from the client application. Ideally, you want this source to return data in JSON format.

Transport Layer
> This source is how you get data from the external source into your internal source, a JSONStore collection inside the store. One alternative is a Worklight adapter.

Internal Data Source API
> This source is the JSONStore APIs that you can use to add JSON data to a collection.

Note: You can populate the internal store with data that is read from a file, an input field, or hardcoded data in a variable. It does not have to come exclusively from an external source that requires network communication.

Example pull scenario

All of the following code examples are written in pseudocode that looks similar to JavaScript.

Note: Use Worklight adapters for the Transport Layer. Some of the advantages of using Worklight adapters are XML to JSON, security, filtering, and decoupling of server-side code and client-side code.

External Data Source: Backend REST endpoint

Imagine that you have a REST endpoint that read data from a database and returns it as an array of JSON objects.

```
app.get('/people', function (req, res) {

  var people = database.getAll('people');

  res.json(people);
});
```

The data that is returned can look like the following example:

```
[{id: 0, name: 'carlos', ssn: '111-22-3333'},
 {id: 1, name: 'mike', ssn: '111-44-3333'},
 {id: 2, name: 'dgonz' ssn: '111-55-3333')]
```

Transport Layer: Worklight adapter

Imagine that you created an adapter that is called `people` and you defined a procedure that is called `getPeople`. The procedure calls the REST endpoint and returns the array of JSON objects to the client. You might want to do more work here, for example, return only a subset of the data to the client.

```
function getPeople () {

  var input = {
    method : 'get',
    path : '/people'
  };

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

On the client, you can use the `WL.Client.invokeProcedure` API to get the data. Additionally, you might want to pass some parameters from the client to the Worklight adapter. One example is a date with the last time that the client got new data from the external source through the Worklight adapter.

```
WL.Client.invokeProcedure({
  adapter : 'people',
  procedure : 'getPeople'.
  parameters : []
})

.then(function (responseFromAdapter) {
  // ...
});
```

Note: You might want to take advantage of the `compressResponse`, `timeout`, and other parameters that can be passed to the `invokeProcedure` API.

Alternatively, you can skip the Worklight adapter and use something like jQuery.ajax to directly contact the REST endpoint with the data that you want to store.

```
$.ajax({
  type: 'GET',
  url: 'http://example.org/people',
```

```
})
.then(function (responseFromEndpoint) {
  // ...
});
```

Internal Data Source API: JSONStore

After you have the response from the backend, you can work with that data by using JSONStore.

JSONStore provides a way to track local changes. It enables some APIs to mark documents as dirty. The API records the last operation that was performed on the document, and when the document was marked as dirty. You can then use this information to implement features like data synchronization.

The change API takes the data and some options:

replaceCriteria

These search fields are part of the input data. They are used to locate documents that are already inside a collection. For example, if you select:

```
['id', 'ssn']
```

as the replace criteria, pass the following array as the input data:

```
[{id: 1, ssn: '111-22-3333', name: 'Carlos'}]
```

and the `people` collection already contains the following document:

```
{_id: 1,json: {id: 1, ssn: '111-22-3333', name: 'Carlitos'}}
```

The `change` operation locates a document that matches exactly the following query:

```
{id: 1, ssn: '111-22-3333'}
```

Then the `change` operation performs a replacement with the input data and the collection contains:

```
{_id: 1, json: {id:1, ssn: '111-22-3333', name: 'Carlos'}}
```

The name was changed from `Carlitos` to `Carlos`. If more than one document matches the replace criteria, then all documents that match are replaced with the respective input data.

addNew

When no documents match the replace criteria, the `change` API looks at the value of this flag. If the flag is set to `true`, the `change` API creates a new document and adds it to the store. Otherwise, no further action is taken.

markDirty

Determines whether the `change` API marks documents that are replaced or added as dirty.

An array of data is returned from the Worklight adapter:

```
.then(function (responseFromAdapter) {

  var accessor = WL.JSONStore.get('people');

  var data = responseFromAdapter.invocationResult.array;

  var changeOptions = {
    replaceCriteria : ['id', 'ssn'],
```

```
      addNew : true,
      markDirty : false
    };

    return accessor.change(data, changeOptions);
})

.then(function() {
   // ...
})
```

You can use other APIs to track changes to the local documents that are
stored. Always get an accessor to the collection that you perform
operations on.

```
var accessor = WL.JSONStore.get('people')
```

Then, you can add data (array of JSON objects) and decide whether you
want it to be marked dirty or not. Typically, you want to set the `markDirty`
flag to `false` when you get changes from the external source. Then, set the
flag to `true` when you add data locally.

```
accessor.add(data, {markDirty: true})
```

You can also replace a document, and opt to mark the document with the
replacements as dirty or not.

```
accessor.replace(doc, {markDirty: true})
```

Similarly, you can remove a document, and opt to mark the removal as
dirty or not. Documents that are removed and marked dirty do not show
up when you use the `find` API. However, they are still inside the collection
until you use the `markClean` API, which physically removes the documents
from the collection. If the document is not marked as dirty, it is physically
removed from the collection.

```
accessor.remove(doc, {markDirty: true})
```

Push

Many systems use the term *push* to refer to sending data to an external source.

There are three important pieces:

Internal Data Source API
> This source is the JSONStore API that returns documents with local-only
> changes (dirty).

Transport Layer
> This source is how you want to contact the external data source to send
> the changes.

External Data Source
> This source is typically a database, REST or SOAP endpoint, among others,
> that receives the updates that the client made to the data.

Example push scenario

All of the following code examples are written in pseudocode that looks similar to
JavaScript.

Note: Use Worklight adapters for the Transport Layer. Some of the advantages of
using Worklight adapters are XML to JSON, security, filtering, and decoupling of
server-side code and client-side code.

Internal Data Source API: JSONStore

After you have an accessor to the collection, you can call the `getAllDirty` API to get all documents that are marked as dirty. These documents have local-only changes that you want to send to the external data source through a transport layer.

```
var accessor = WL.JSONStore.get('people');

accessor.getAllDirty()

.then(function (dirtyDocs) {
  // ...
});
```

The `dirtyDocs` argument looks like the following example:

```
[{_id: 1,
  json: {id: 1, ssn: '111-22-3333', name: 'Carlos'},
  _operation: 'add',
  _dirty: '1395774961,12902'}]
```

The fields are:

_id
> Internal field that JSONStore uses. Every document is assigned a unique one.

json
> The data that was stored.

_operation
> The last operation that was performed on the document. Possible values are `add`, `store`, `replace`, and `remove`.

_dirty
> A time stamp that is stored as a number to represent when the document was marked dirty.

Transport Layer: Worklight adapter

You can choose to send dirty documents to a Worklight adapter. Assume that you have a `people` adapter that is defined with an `updatePeople` procedure.

```
.then(function (dirtyDocs) {

  return WL.Client.invokeProcedure({
    adapter : 'people',
    procedure : 'updatePeople',
    parameters : [ dirtyDocs ]
  });
})

.then(function (responseFromAdapter) {
  // ...
})
```

Note: You might want to take advantage of the `compressResponse`, `timeout`, and other parameters that can be passed to the `invokeProcedure` API. On the Worklight Server, the adapter has the `updatePeople` procedure, which might look like the following example:

```
function updatePeople (dirtyDocs) {

  var input = {
    method : 'post',
    path : '/people',
```

```
        body: {
          contentType : 'application/json',
          content : JSON.stringify(dirtyDocs)
        }
    };

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

Instead of relaying the output from the `getAllDirty` API on the client, you might have to update the payload to match a format that is expected by the backend. You might have to split the replacements, removals, and inclusions into separate backend API calls.

Alternatively, you can iterate over the `dirtyDocs` array and check the `_operation` field. Then, send replacements to one procedure, removals to another procedure, and inclusions to another procedure. The previous example sends all dirty documents in bulk to the Worklight adapter.

```
var len = dirtyDocs.length;
var arrayOfPromises = [];

while (len--) {

  var currentDirtyDoc = dirtyDocs[len];

  switch (currentDirtyDoc._operation) {

    case 'add':
    case 'store':

      arrayOfPromises.push(WL.Client.invokeProcedure({
        adapter : 'people',
        procedure : 'addPerson',
        parameters : [ currentDirtyDoc ]
      }));

    break;

    case 'replace':
    case 'refresh':

      arrayOfPromises.push(WL.Client.invokeProcedure({
        adapter : 'people',
        procedure : 'replacePerson',
        parameters : [ currentDirtyDoc ]
      }));

    break;

    case 'remove':
    case 'erase':

      arrayOfPromises.push(WL.Client.invokeProcedure({
        adapter : 'people',
        procedure : 'removePerson',
        parameters : [ currentDirtyDoc ]
      }));
  }
}

$.when.apply(this, arrayOfPromises)
.then(function () {
  var len = arguments.length;
```

```
      while (len--) {
        // Look at the responses in arguments[len]
      }
    });
```

Alternatively, you can skip the Worklight adapter and contact the REST endpoint directly.

```
.then(function (dirtyDocs) {

  return $.ajax({
    type: 'POST',
    url: 'http://example.org/updatePeople',
    data: dirtyDocs
  });
})

.then(function (responseFromEndpoint) {
  // ...
});
```

External Data Source: Backend REST endpoint

The backend accepts or rejects changes, and then relays a response back to the client. After the client looks at the response, it can pass documents that were updated to the `markClean` API.

```
.then(function (responseFromAdapter) {

  if (responseFromAdapter is successful) {
    WL.JSONStore.get('people').markClean(dirtyDocs);
  }
})

.then(function () {
  // ...
})
```

After documents are marked as clean, they do not show up in the output from the `getAllDirty` API.

JSONStore wizard (JavaScript only)

The Worklight JSONStore wizard can help you create a template JavaScript file that is based on search fields that are selected from the backend that you provide. Using the wizard is optional.

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 that is 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.

JSONStore analytics

You can enable the collection of analytics information for Android and iOS.

Overview

You can collect key pieces of analytics information that are related to JSONStore with the Worklight platform.

File information
> File information is collected once per application session if the JSONStore API is called with the analytics flag set to `true`. An application session is defined as loading the application into memory and removing it from memory. You can use this information to determine how much space is being used by JSONStore content in the application.

Performance metrics
> Performance metrics are collected every time a JSONStore API is called with information about the start and end times of an operation. You can use this information to determine how much time various operations take in milliseconds.

Examples

iOS

```
JSONStoreOpenOptions* options = [JSONStoreOpenOptions new];
[options setAnalytics:YES];

[[JSONStore sharedInstance] openCollections:@[...] withOptions:options error:nil];
```

Android

```
JSONStoreInitOptions initOptions = new JSONStoreInitOptions();
initOptions.setAnalytics(true);

WLJSONStore.getInstance(...).openCollections(..., initOptions);
```

JavaScript

This example applies only when the application is running on the Android or iOS environments.

```
var options = {
  analytics : true
};

WL.JSONStore.init(..., options);
```

JSONStore security utilities

Learn about JSONStore security utilities.

JSONStore security utilities overview

The Worklight client-side API provides some security utilities to help protect your user's data. Features like JSONStore are great if you want to protect JSON objects. However, it is not recommended to store binary blobs in a JSONStore collection.

Instead, store binary data on the file system, and store the file paths and other metadata inside a JSONStore collection. If you want to protect files like images, you can encode them as base64 strings, encrypt it, and write the output to disk. When it is time to decrypt the data, you can look up the metadata in a JSONStore collection, read the encrypted data from the disk, and decrypt it using the metadata that was stored. This metadata can include the key, salt, Initialization Vector (IV), type of file, path to the file, and others.

At a high level, the SecurityUtils API provides the following APIs:

- Key generation - Instead of passing a password directly to the encryption function, this key generation function uses Password Based Key Derivation Function v2 (PBKDF2) to generate a strong 256-bit key for the encryption API. It takes a parameter for the number of iterations. The higher the number, the more time it takes an attacker to brute force your key. Use a value of at least 10,000. The salt must be unique and it helps ensure that attackers have a harder time using existing hash information to attack your password. Use a length of 32 bytes.
- Encryption - Input is encrypted by using the Advanced Encryption Standard (AES). The API takes a key that is generated with the key generation API. Internally, it generates a secure IV, which is used to add randomization to the first block cipher. Text is encrypted. If you want to encrypt an image or other binary format, turn your binary into base64 text by using these APIs. This encryption function returns an object with the following parts:
  - ct (cipher text, which is also called the encrypted text)
  - IV
  - v (version, which allows the API to evolve while still being compatible with an earlier version)
- Decryption - Takes the output from the encryption API as input, and decrypts the cipher or encrypted text into plain text.

- Remote random string - Gets a random hex string by contacting a random generator on the Worklight Server. The default value is 20 bytes, but you can change the number up to 64 bytes.
- Local random string - Gets a random hex string by generating one locally, unlike the remote random string API, which requires network access. The default value is 32 bytes and there is not a maximum value. The operation time is proportional to the number of bytes.
- Encode base64 - Takes a string and applies base64 encoding. Incurring a base64 encoding by the nature of the algorithm means that the size of the data is increased by approximately 1.37 times the original size.
- Decode base64 - Takes a base64 encoded string and applies base64 decoding.

JSONStore security utilities setup

Ensure that you import the following files to use the JSONStore security utilities APIs.

iOS

```
#import "WLSecurityUtils.h"
```

Android

```
import com.worklight.wlclient.api.SecurityUtils
```

JavaScript

No setup is required.

JSONStore security utilities examples

Learn about JSONStore security utilities examples.

JSONStore security utilities iOS examples:

Learn about JSONStore security utilities iOS examples.

Encryption and decryption

```
// User provided password, hardcoded only for simplicity.
NSString* password = @"HelloPassword";

// Random salt with recommended length.
NSString* salt = [WLSecurityUtils generateRandomStringWithBytes:32];

// Recomended number of iterations.
int iterations = 10000;

// Populated with an error if one occurs.
NSError* error = nil;

// Call that generates the key.
NSString* key = [WLSecurityUtils generateKeyWithPassword:password
                                 andSalt:salt
                                 andIterations:iterations
                                 error:&error];

// Text that is encrypted.
NSString* originalString = @"My secret text";
NSDictionary* dict = [WLSecurityUtils encryptText:originalString
                                 withKey:key
                                 error:&error];

// Should return: 'My secret text'.
NSString* decryptedString = [WLSecurityUtils decryptWithKey:key
                                        andDictionary:dict
                                        error:&error];
```

Encode and decode base64

```
// Input string.
NSString* originalString = @"Hello world!";

// Encode to base64.
NSData* originalStringData = [originalString dataUsingEncoding:NSUTF8StringEncoding];
NSString* encodedString = [WLSecurityUtils base64StringFromData:originalStringData length:originalString.length];

// Should return: 'Hello world!'.
NSString* decodedString = [[NSString alloc] initWithData:[WLSecurityUtils base64DataFromString:encodedString] encoding:NSUTF8StringEr
```

Get remote random

```
[WLSecurityUtils getRandomStringFromServerWithBytes:32
                timeout:1000
                completionHandler:^(NSURLResponse *response, NSData *data, NSError *connectionError) {

  // You might want to see the response and the connection error before moving forward.

  // Get the secure random string.
  NSString* secureRandom = [[NSString alloc] initWithData:data encoding:NSUTF8StringEncoding];
}];
```

Get local random

```
NSString* secureRandom = [WLSecurityUtils generateRandomStringWithBytes:32];
```

JSONStore security utilities Android examples:

Learn about JSONStore security utilities Android examples.

Encryption and decryption

```
String password = "HelloPassword";
String salt = SecurityUtils.getRandomString(32);
int iterations = 10000;

String key = SecurityUtils.generateKey(password, salt, iterations);

String originalText = "Hello World!";

JSONObject encryptedObject = SecurityUtils.encrypt(key, originalText);

// Deciphered text will be the same as the original text.
String decipheredText = SecurityUtils.decrypt(key, encryptedObject);
```

Encode and decode base64

```
import android.util.Base64;

String originalText = "Hello World";
byte[] base64Encoded = Base64.encode(text.getBytes("UTF-8"), Base64.DEFAULT);

String encodedText = new String(base64Encoded, "UTF-8");

byte[] base64Decoded = Base64.decode(text.getBytes("UTF-8"), Base64.DEFAULT);

// Decoded text will be the same as the original text.
String decodedText = new String(base64Decoded, "UTF-8");
```

Get remote random

```
Context context; // This is the current Activity's context.
int byteLength = 32;

// Listener calls the callback functions after it gets the response from the server.
WLRequestListener listener = new WLRequestListener(){
  @Override
  public void onSuccess(WLResponse wlResponse) {
    // Implement the success handler.
  }

  @Override
  public void onFailure(WLFailResponse wlFailResponse) {
    // Implement the failure handler.
    }
};

SecurityUtils.getRandomStringFromServer(byteLength, context, listener);
```

Get local random

```
int byteLength = 32;
String randomString = SecurityUtils.getRandomString(byteLength);
```

JSONStore security utilities JavaScript examples:

Learn about JSONStore security utilities JavaScript examples.

Encryption and decryption

```
// Keep the key in a variable so that it can be passed to the encrypt and decrypt API.
var key;

// Generate a key.
WL.SecurityUtils.keygen({
  password: 'HelloPassword',
  salt: Math.random().toString(),
  iterations: 10000
})

.then(function (res) {

  // Update the key variable.
  key = res;

  // Encrypt text.
  return WL.SecurityUtils.encrypt({
    key: key,
    text: 'My secret text'
  });
})

.then(function (res) {

  // Append the key to the result object from encrypt.
  res.key = key;

  // Decrypt.
  return WL.SecurityUtils.decrypt(res);
})

.then(function (res) {

  // Remove the key from memory.
  key = null;

  //res => 'My secret text'
})

.fail(function (err) {
  // Handle failure in any of the previously called APIs.
});
```

Encode and decode base64

```
WL.SecurityUtils.base64Encode('Hello World!')
.then(function (res) {
  return WL.SecurityUtils.base64Decode(res);
})
.then(function (res) {
  //res => 'Hello World!'
})
.fail(function (err) {
  // Handle failure.
});
```

Get remote random

```
WL.SecurityUtils.remoteRandomString(32)
.then(function (res) {
  // res => deba58e9601d24380dce7dda85534c43f0b52c342ceb860390e15a638baecc7b
})
.fail(function (err) {
  // Handle failure.
});
```

Get local random

```
WL.SecurityUtils.localRandomString(32)
.then(function (res) {
  // res => 40617812588cf3ddc1d1ad0320a907a7b62ec0abee0cc8c0dc2de0e24392843c
})
.fail(function (err) {
  // Handle failure.
});
```

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 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 Worklight makes the entire process of communicating with the users and devices completely transparent to the developer.
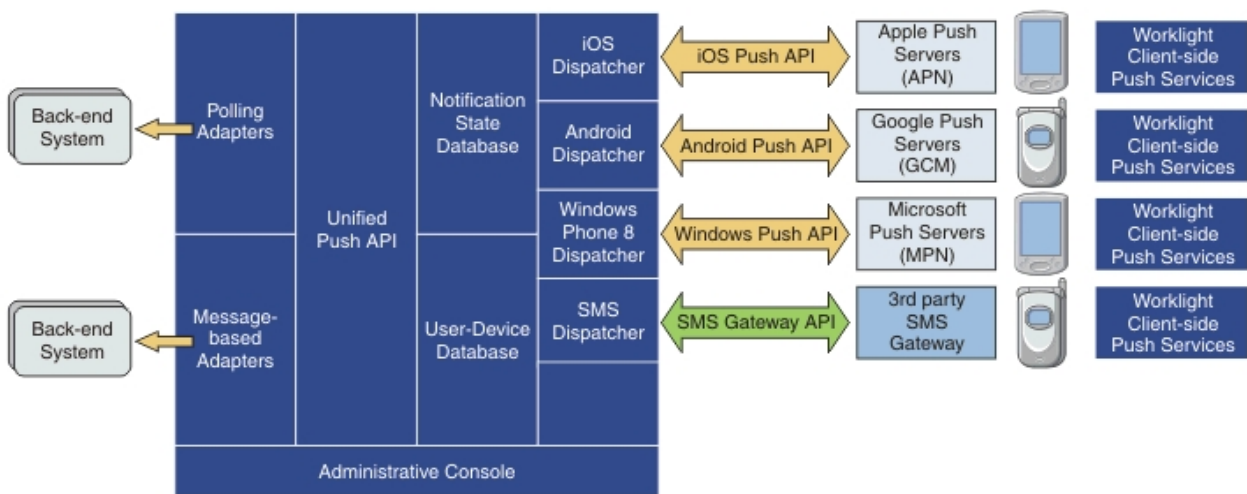
Figure 105. Push notification mechanism

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 Worklight applications on the server" on page 1171.

Architecture

Unlike other IBM Worklight Foundation services, the push server requires outbound connections to Apple, Google, and Microsoft servers using ports that are defined by these companies.

For more information, see "Possible Worklight push notification architectures."

Possible Worklight push notification architectures

IBM Worklight Foundation supports two different methods of implementing push notifications, which are based on how the enterprise back end provides the messages to the Worklight Server.

Two common ways exist to create an IBM Worklight Foundation push notification architecture:
- The Java Message Service (JMS) polling method, in which messages are pulled from the JMS message queue and sent by the Worklight Server
- The enterprise back end method, in which an enterprise back end uses a Worklight adapter to deliver messages to a Worklight Server cluster

JMS polling architecture

This architecture relies on the enterprise backend to deliver messages to a single instance of Worklight Server by using a JMS message queue. The developer must create an IBM Worklight Foundation JMS adapter, which pulls messages from the queue and calls the IBM Worklight Foundation server-side push notification API to process the messages.
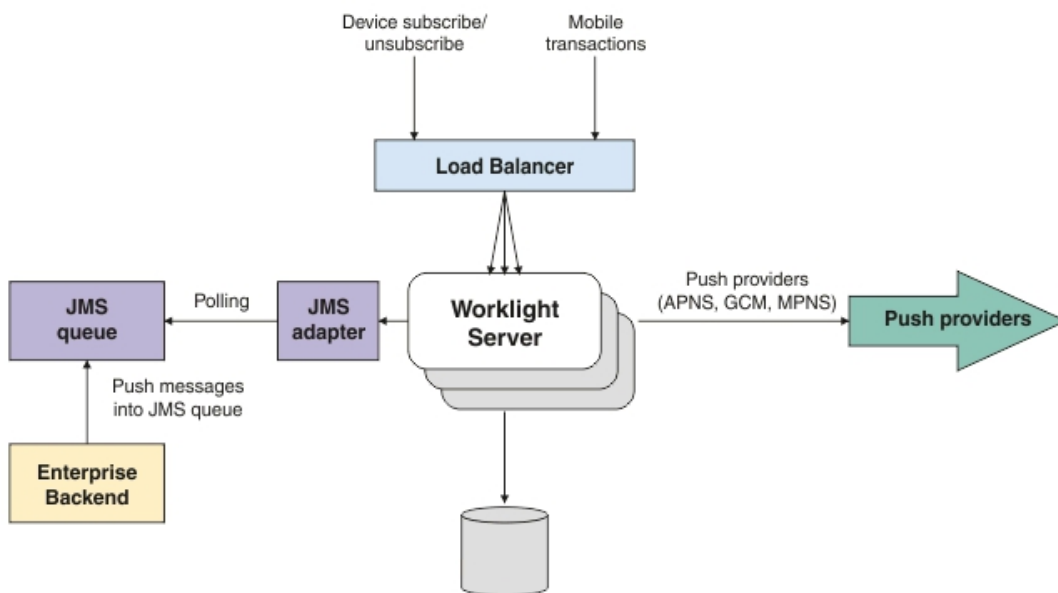
Figure 106. JMS polling push notification architecture

When this architecture is used, the flow is as follows:

1. Messages are put into the JMS queue by the enterprise backend.
2. The Worklight Server polls the JMS queue by using the JMS adapter, retrieving messages in short batches and sending them to the push providers.
3. A single Worklight Server instance pulls from the JMS queue and sends the push notifications. Even in a Worklight Server cluster, only one Worklight Server polls.
4. The process is implemented by using a Worklight JMS adapter, which functions as follows:
   - In a Worklight Server cluster, the single polling Worklight Server is selected randomly, by using the IBM Worklight Foundation cluster-sync mechanism.
   - If the server that pulls from the JMS queue is shut down, another server takes its place.

This is the standard architecture. *Pros* of this method are that it involves an asynchronous queue, into which you can put the messages that you want to send. These messages are then processed and pulled later by the Worklight Server. *Cons* of this method are that only one server is sending the push notifications, so the maximum messages-per-second throughput is fixed.

Enterprise backend calling the Worklight Server architecture

This architecture relies on the enterprise backend to deliver messages to a Worklight Server cluster by calling a Worklight adapter procedure.
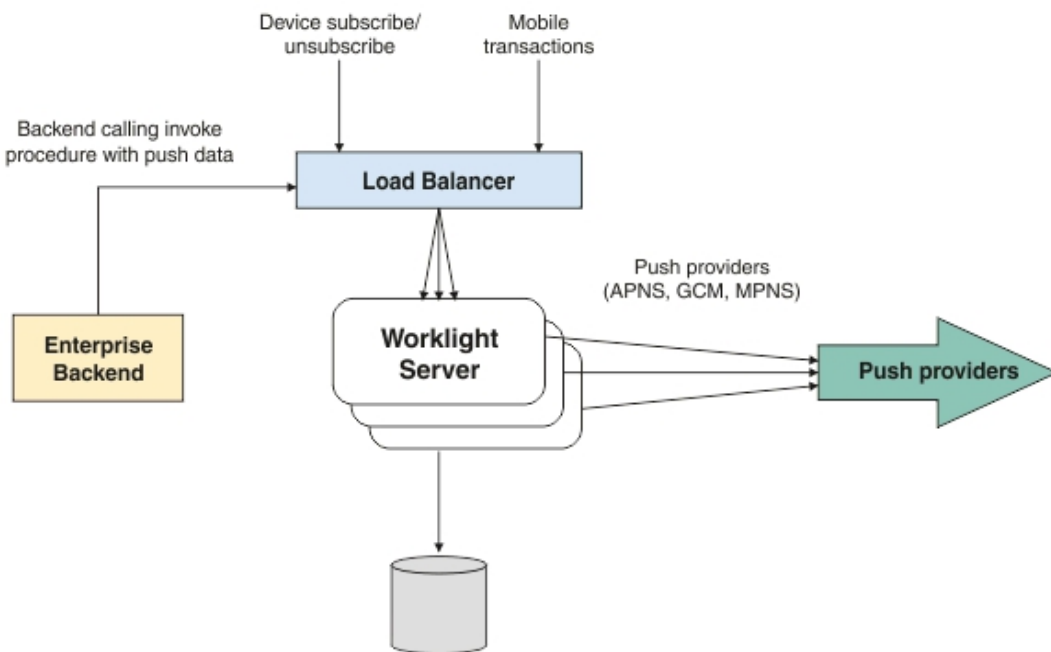
Figure 107. Enterprise backend push notification architecture

When this architecture is used, the flow is as follows:
1. The request is routed to one of the Worklight Server instances, which sends a push message to a provider.
2. In this flow, all Worklight Server instances can send push notifications, but for a specific request only one of the server instances performs the task.

3. The enterprise backend initiates calls to the load balancer.

Pros of this method are that all Worklight Server can be used to send push notifications, so you can add more servers if you must send more messages per second. *Cons* of this method are that every push message is a transaction on the Worklight Server. You can mitigate this overhead by sending a number of messages together or by having the Worklight adapter procedure that is invoked call the backend for a batch of messages rather than single messages.

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 server key. The generated key is the `key` value that you use in the `application-descriptor.xml` file. When you create the server key, do not restrict it to any specific URL. For more information about how to create the key, see Obtaining an API key.
2. If your organization has a firewall that restricts the traffic to or from the Internet, you must do the following steps:
   a. Configure the firewall to allow connectivity with GCM in order for your GCM client apps to receive messages. The ports to open are 5228, 5229, and 5230. GCM typically uses only 5228, but it sometimes uses 5229 and 5230. GCM does not provide specific IP, so you must allow your firewall to accept outgoing connections to all IP addresses contained in the IP blocks listed in Google's ASN of 15169. For more information, see Implementing an HTTP Connection Server.
   b. Ensure that your firewall accepts outgoing connections from Worklight Server to `android.googleapis.com` on port 443.
3. 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). 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. Follow the required steps to obtain your APNS certificate and password. For more 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 by using SSL port 443.

   While you work in development mode, rename your certificate file to `apns-certificate-sandbox.p12`. When you move to production, rename your certificate file to `apns-certificate-production.p12`. In both cases, place the certificate file in the environment root folder or in the application root folder. When the hybrid application has both iPhone and iPad environments, separate certificates are necessary for push notification. In that case, place those certificates in the corresponding environment folders.

   Note: The environment root folder takes the highest priority.

   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 notification 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 by Worklight® Server to extract the certificate from the Java™ keystore file, so that it can be used in the handshake process with Microsoft Push Notification Service (MPNS). Any push notifications submitted to the application will be authenticated and secure.

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>
```

Broadcast notifications

Broadcast notifications are notification messages that are targeted to all the devices that have the Worklight application installed and configured for push notifications.

Broadcast notifications are enabled by default with any Worklight application that is enabled for push notification. For more information about configuring your application for push notifications, see "Setting up push notifications" on page 844.

Any Worklight application that is enabled for push notification has a predefined subscription to the `Push.ALL` tag, which is used by Worklight Server to broadcast notification messages to all the devices.

To disable broadcast notification for native app, use `unsubscribeTag` method of WLPush class, with the tag name `Push.ALL`. If you want to disable broadcast notification for hybrid app, use the `unsubscribeTag` method of the WL.Client.Push class, with the tag name `Push.ALL`.

Note: Make sure that the `WL.Client.Push.onMessage` method is defined in the application that is called when the push notification arrives.

For more information about sending broadcast notification, see "Broadcast notification" on page 852 section in "Sending push notifications" on page 852.

Event source-based notifications

Event source-based notifications are notification messages that are targeted to devices with a user subscription.

An event source can either poll notifications from the backend system, or wait for the backend system to explicitly push a new notification. The process of the event source-based notifications is as follows: :

1. Notifications are retrieved by the Worklight adapter event source, either by poll or by push from the backend system.
2. The adapter processes the notification and sends it to an Apple, Google, or Microsoft push service mediator.
3. The push service mediator sends a push notification to the device.
4. The device processes the received notification.

To start receiving push notifications, an application must subscribe to an event source. The event source is a push notification channel to which mobile applications can register. You can create an event source by declaring a notification event source in the Worklight adapter JavaScript code at a global level, outside any JavaScript function. For example,

```
WL.Server.createEventSource({
  name: 'PushEventSource',
  onDeviceSubscribe: 'deviceSubscribeFunc',
  onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
  securityTest:'PushApplication-strong-mobile-securityTest'
});
```

For more information about createEventSource method, see WL.Server class.

After the creation of the event source, you can proceed with the push notification subscriptions that is described in "Subscribing to an event source."

For more information about sending event source-based notification, see Event source-based notification section in "Sending push notifications" on page 852.

Subscribing to an event source

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 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.

To set up the device subscription for hybrid application, use the methods of the WL.Client.Push class. For native application, use the methods of WLPush class.

Procedure

1. When the application first connects to the Worklight Server from the device, the device registers with a push service mediator and obtains a device token. This process is done automatically by IBM Worklight Foundation.
2. When the token is obtained, the `onReadyToSubscribe` callback function that is defined in the application is notified that a device is ready to subscribe to push notifications.
3. After the `onReadyToSubscribe` callback is notified, the application subscribes to a tag by using the `subscribe` API.
4. Optional: If push notifications are no longer required, you can unsubscribe. The device subscription is deleted either by an application that calls the `unsubscribe` API, or when the push mediator informs the Worklight Server that the device is permanently inaccessible.

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 subscribed from.

What to do next

The event source-based notifications can be delivered by the adapter code to all or some of the devices that the user subscribed from. For more information, see Event source-based notification section in "Sending push notifications" on page 852.

Tag-based notification

Tag notifications are notification messages that are targeted to all the devices that are subscribed to a particular tag.

Tags-based notifications allow segmentation of notifications based on subject areas or topics. Notification recipients can choose to receive notifications only if it is about a subject or topic that is of interest. Therefore, tags-based notification provides a means to segment recipients. This feature enables ability to define tags and then send and receive messages by tags. A message is targeted to only the devices that are subscribed to a tag.

You must first create the tags for the application, set up the tag subscriptions and then initiate the tag-based notifications. For more information, see "Setting up Tag-based notifications."

For more information about sending tag-based notification, see "Tag-based notification" on page 853 section in "Sending push notifications" on page 852.

Setting up Tag-based notifications

You create tags for an application by specifying tag details in the `application-descriptor.xml` file. You can then set up tag subscriptions and initiate tag-based notifications.

Before you begin

In the `application-descriptor.xml` file, add a `<tags>` element to specify the tags that you want for the application, such as customer categories.

The `<tags>` element is supported by Android, iOS, and Windows Phone 8 environments. Tags represent topics of interest to the user and provide user with an ability to receive notifications according to the chosen interest. During application deployment, the specified tags are created, updated, or deleted on the management database tables. This feature enables ability for sending and receiving messages by tags. A message is targeted to only the devices that are subscribed for a tag. In the following example, the `<tags>` specifies customer categories.

```
<tags>
  <tag>
    <name>Silver</name>
    <description>Silver customers</description>
  </tag>
  <tag>
    <name>Gold</name>
    <description>Gold customers</description>
  </tag>
</tags>
```

About this task

To receive notifications that are targeted to a particular tag, you subscribe the application to the tags that you defined. To set up tag subscriptions for hybrid application, you use the methods of the WL.Client.Push class. For native

application, use the methods of WLPush class.

Procedure

1. When the application first connects to Worklight Server from the device, the device registers with a push service mediator and obtains a device token. This process is done automatically by IBM Worklight Foundation.

2. When the token is obtained, the onReadyToSubscribe callback method that is defined in the application is notified that a device is ready to subscribe to push notifications.

3. After the onReadyToSubscribe callback is notified, the application subscribes to a tag by calling the subscribeTag method.

4. Optional: You can unsubscribe the application in one of two ways:
   - The device subscription is deleted by an application that calls the unsubscribeTag method.
   - The push mediator informs Worklight Server that the device is permanently unaccessible.

5. Make sure that the onMessage method is defined in the application that is called when the push notification arrives. For hybrid application, see the WL.Client.Push class. For Android and Windows Phone 8 native application, see the WLNotificationListener class. The implementation for iOS native application is different. The notification comes to didReceiveRemoteNotification method that is defined by default in your appDelegate.m file.

Results

While the tag subscription exists, Worklight Server can produce push notifications for the subscribed tag. These notifications can be delivered by the adapter code to all the devices that subscribed to the tag.

You can view the notification tags in the Worklight Console.

Last deployed at: 4/9/2014 4:35 PM

Push tag(s): OFFER10, PLATINUM, OFFER20, SILVER, GOLD, BRONZE
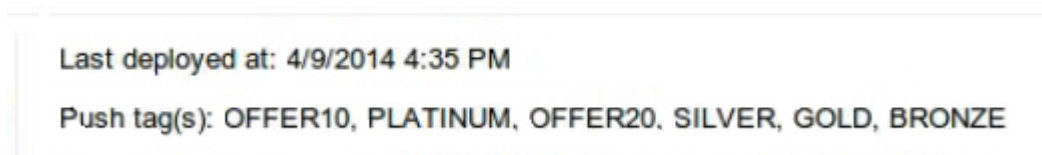
What to do next

After you have set up tag subscriptions, you can send a notification. For more information, see "Tag-based notification" on page 853 section in "Sending push notifications" on page 852.

Unicast notifications

Unicast notifications are notification messages that are targeted to a particular device or a userID.

Unicast notifications do not require any additional setup and are enabled by default when the Worklight application is enabled for push notifications. For more information about configuring your application for push notifications, see "Setting up push notifications" on page 844.

For more information about sending unicast notification, see "Unicast notification" on page 853 section in "Sending push notifications" on page 852.

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 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 142. 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 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 Foundation 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, 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

When you have set up push notification, whether event-source based, tag-based, or broadcast-enabled, you can send push notifications from the server.

Broadcast notification

Before you can send a broadcast notification, you must set up broadcast notifications for the required applications. For more information, see "Broadcast notifications" on page 847.

You can send a broadcast notification in the following way:
- Use the `sendMessage` method of the WL.Server class. The **applicationId** and **notificationOptions** parameters are mandatory.
- The **notificationOptions.target** object must not be specified or empty.

Event source-based notification

Before you can send event source-based notification, you must set up the subscriptions. For more information, see "Subscribing to an event source" on page 848.

An event source can either poll notifications from the backend system, or wait for the backend system to explicitly push a new notification. In this example, a `submitNotifications()` adapter function is called by a backend system as an external API to send notifications.

```
function submitNotification(userId, notificationText) {
  varuserSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource', u
  if(userSubscription === null) {
    return{ result: "No subscription found for user :: "+ userId };
  }

  varbadgeDigit = 1;
  varnotification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"dat
    WL.Server.notifyAllDevices(userSubscription, notification);
  return{
    result: "Notification sent to user :: "+ userId
  };
}
```

For more information about the various APIs to send notifications, see WL.Server.

Tag-based notification

Before you can send tag-based notifications, you must set up tag subscriptions. For more information, see "Setting up Tag-based notifications" on page 849.

You can send a tag-based notification in the following way:
- Use the `sendMessage` method of the WL.Server class. The **applicationId** and **notificationOptions** parameters are mandatory.
- Specify the **tagNames** as an array in the **notificationOptions.target.tagNames** object.

Unicast notification

You can send a unicast notification to a particular device in the following way:
- Use the `sendMessage` method of the WL.Server class. The **applicationId** and **notificationOptions** parameters are mandatory.
- The **deviceId(s)** as an array in the **notificationOptions.target.deviceIds** object.

You can send a unicast notification to a particular user in the following way:
- Use the `sendMessage` method of the WL.Server class. The **applicationId** and **notificationOptions** parameters are mandatory.
- The **userId(s)** as an array in the **notificationOptions.target.userIds** object.

Note: The notification message can target multiple devices or users by specifying multiple **deviceIDs** or **userIDs** in the **notificationOptions.tager.deviceIds** or **notificationOptions.target.userIds**.

Platform or environment-based notification

You can send a platform or environment-based notification in the following way:

- Use the `sendMessage` method of the WL.Server class. The **applicationId** and **notificationOptions** parameters are mandatory.
- Specify the platform or platforms as an array in the **notificationOptions.target.platforms** object. The supported platforms are:
  - A (Apple)
  - G (Google)
  - M (Microsoft)

Restriction

Restriction: The `sendMessage` method does not support SMS notification. For more information, see "Sending SMS push notifications."

Sending SMS push notifications

In addition to standard push notifications, you can also send Short Message Service (SMS) messages, more commonly known as text messages, to user devices. To receive SMS notifications, users must first subscribe to a push notification event source.

About this task

The SMS notification framework extends the push notification framework. SMS support is provided for Apple, Google, and Windows Phone 8 devices, and for BlackBerry devices that support SMS functions. IBM Worklight Foundation includes the capability to send SMS notifications to all platforms that provide SMS support.

Procedure

1. An SMS notification infrastructure is set up.

   A Worklight adapter acts as a connector to an app that is running on a mobile device.
2. The user of the mobile device sends a subscribe request from the application to the event source that is declared in the Worklight adapter, by using the client-side `WL.Client.Push.subscribeSMS` method.
3. The user subscription to the event source is registered at the Worklight Server.
4. When the back-end service must notify the user, it calls a method in the Worklight adapter.
5. The adapter checks whether an SMS subscription 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 necessary, you can unsubscribe.

   The subscription is deleted either by an application that calls the `WL.Client.Push.unsubscribeSMS` method, or by using the Admin console. For more information, see Administering push notifications with the Worklight Console.

   For a detailed scenario-based example that shows 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 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:

  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:

  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:

  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 that Worklight adapters 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 Worklight notification framework. For more information, see Push notification.

The two-way SMS architecture is summarized in the following figure:
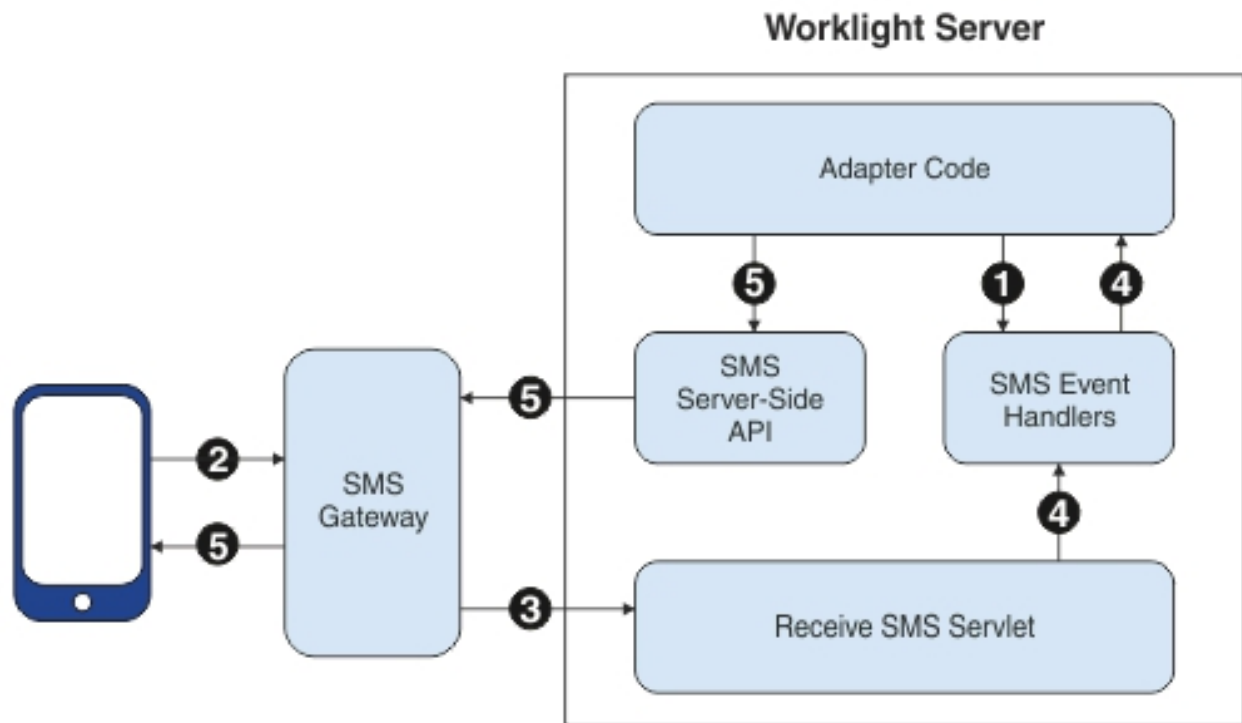
Figure 108. Two-way SMS architecture

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.

Using native and JavaScript push APIs in the same app

You can use Native and JavaScript push APIs in the same Worklight app.

Overview

The Worklight Framework provides two sets of APIs to handle push notifications - native and web (JavaScript). Each of these APIs on its own allows the application to handle push notifications in its native/JavaScript environment. However, you can also use both API types in the same app, provided you follow certain restrictions:

iOS   Both APIs rely on an automatic process to initialize push support, triggered when connecting to the Worklight Server. In case the first connection to the server is made from native code, the Native API must be used for the Push management operations throughout the app lifecycle (subscribe, unsubscribe, registerEventSourceCallback, and others). If the first connection is made form the JavaScript code, the JavaScript API must be used for these operations.

Android
In the Android Environment, there is a stronger restriction. The native Push API cannot be used in a hybrid application. The JavaScript API must be used for Push management operations (subscribe, unsubscribe, and others).

Reacting to push notifications

It is possible to react to push notifications both in native and JavaScript code in the same app. Using both provides an enhanced user experience because it allows the app to react to push notifications in the appropriate context. To achieve the use of both, follow these steps:

Using a native API for managing push operations (iOS only)
If you are coding in native iOS to handle the received push, forward the data to JavaScript using the Action Sender API.

Here is an example of JavaScript code:

```
// register an action receiver function
WL.App.addActionReceiver("myActionReceiverID", myActionReceiver);

// define action receiver function
function myActionReceiver(received)
{
   if (received.action == "handlePushFromNative")
```

```
{
  // handle notification - do something with received.data
}
}
```

In iOS native code, push notification is handled by the following method:

```
(void)application:(UIApplication*)application didReceiveRemoteNotification:(NSDictionary*)
```

in the implementation of the `MyAppDelegate` interface. For example:

```
(void)application:(UIApplication*)applicationdidReceiveRemoteNotification:(NSDictionary*)u
{
  BOOL shouldHandleInNative = some applicative logic;
    if (shouldHandleInNative)
    {
      // handle notification in native code
    }
    else
    {
      // send the push data to a Javascript action receiver
        [[WL sharedInstance]sendActionToJS:@"handlePushFromNative" withData:userInfo];
    }
}
```

Using a JavaScript API for managing push operations

When the JavaScript API is used to handle push, the push notifications are handled by the JavaScript callback function that is registered using the API function `WL.Client.Push.registerEventSourceCallback (alias, adapter, eventSource, callback)`. To handle the notifications using the native code of the app, code as follows:

In JavaScript code

Define and register a callback handler for the push notifications. For example:

```
WL.Client.Push.registerEventSourceCallback(
  "myPush",
  "PushAdapter",
  "PushEventSource",
  pushNotificationReceived
);

function pushNotificationReceived(props, payload) {
  alert("pushNotificationReceived invoked");
  alert("props :: " + JSON.stringify(props));
  alert("payload :: " + JSON.stringify(payload));
}
```

In native Android code

1. Add a default constructor to the `GCMIntentService.java` class. For example:

   ```
   public GCMIntentService(){
     super();

     setBroadcastReceiver(new BroadcastReceiver() {

       @Override
       public void onReceive(Context context, Intent intent) {
         //This is Empty Broadcast Receiver, do not implement any logic here
       }
     });
   }
   ```

2. In a custom native activity, implement the logic for handling the received push notification. Define a new broadcast receiver. For example, you can implement it in the `onStart` method of the native activity, as follows:

```
@Override
protected void onStart() {
  super.onStart();

  //Action name for inner push messages
  String action = getPackageName() + "." + getString(R.string.app_name) + ".C2DM_MESSAGE"

  //Register Custom Push Broadcast Receiver
  registerReceiver(new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {

      if (isHandleInNative){ //Some variable controlled by applicative code
      Message message = intent.getParcelableExtra("message");

      // Custom Native Logic here using the Message class, use message.toString()
      // to receive the JSON object with push notification data.

      //Abort will prevent from sending the broadcast to hybrid part
      abortBroadcast();
      }
    }
  }, new IntentFilter(action));
}
```

In native iOS code Add the method

```
-(void)application:(UIApplication*)application didReceiveRemoteNotification:(NSDictionary*)us
```

to the implementation of the `MyAppDelegate` interface. The method should implement the following logic:

```
- (void)application:(UIApplication*)application didReceiveRemoteNotification:(NSDictionary*)
{
  BOOL shouldHandleInNative = some applicative logic;
  if (shouldHandleInNative)
  {
    // handle notification
  }
  else
  {
    // call the worklight framework implementation - this will
    // invoke the Javascript callback handler
    [super application:application didReceiveRemoteNotification:userInfo];
  }
}
```

Worklight security framework

This collection of topics contains information about and tasks for using Worklight security framework in applications.

Worklight security overview

An overview of security features within IBM Worklight Foundation.

The following sections provide high-level information about the Worklight security model.

Goals and structure of Worklight security framework

The 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 IBM Worklight Foundation 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 IBM Worklight Foundation 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.

Worklight protocol
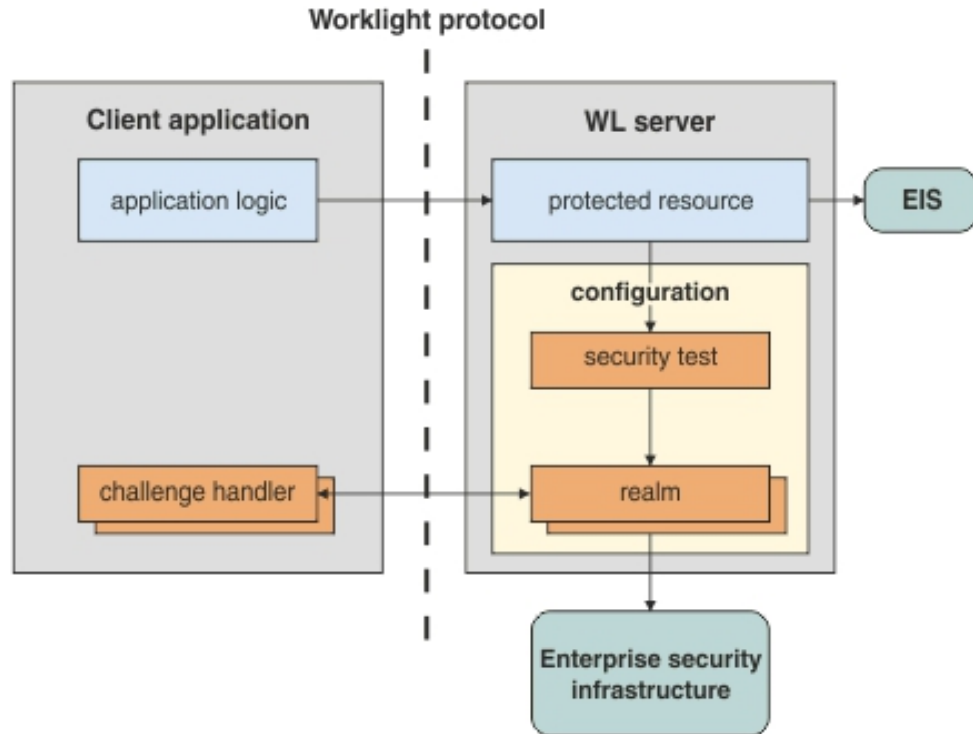
Protected resources and authentication context

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 realms are relevant only to mobile or only to web environments, the configuration of a security test can be non-trivial. IBM Worklight Foundation 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 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 the challenge handler 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. Since this task can be non-trivial, it is important to understand the differences between IBM Worklight Foundation and Web Container authentication models:

- The Java Platform, Enterprise Edition model allows only one authentication scheme for a web application. Multiple resource collections are defined by URL patterns, with authentication constraints defined by a white list of role names.
- The Worklight model, by contrast, 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 IBM Worklight Foundation 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 Foundation, 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 Foundation 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 IBM Worklight Foundation and application authentication.

For more information about IBM Security Access Manager, see IBM Security Access Manager for IBM Worklight.

Worklight application authenticity overview

An overview of application authenticity features and procedures within IBM Worklight Foundation

IBM Worklight Foundation framework provides a number of security mechanisms. One of them is a security test for application authenticity. Most Worklight security mechanisms are based on the same concept: obtaining identity through challenge handling. Just as the user authentication realm is used to obtain and validate the identity of a user, an application authenticity realm is used to obtain and validate the identity of an application. Therefore, this process is referred to as *application authenticity*.

Any entity can access HTTP services (APIs) that are available from Worklight Server by issuing an HTTP request. Therefore, 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 Worklight Server is authentic and was not tampered with or modified by some attacker.

Note: This authenticity feature is not available if you use the internal Worklight Development Server that is embedded in Worklight Studio. For more information, see "Installing Worklight Studio" on page 58.

Authenticity process

Application authenticity checks use 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 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 following ones:
- The token is not processed by JavaScript; instead it is processed with compiled native code. This procedure ensures that no attacker can see the logic behind the token processing.
- Application authenticity is based on certificate keys, which are used to sign the application bundle. Only the developer or enterprise who has access to the original private key of the application can to modify, repackage, and resign the bundle. This process ensures tight security.

Enabling an application authenticity check (example)

Currently, application authenticity is supported only on iOS and Android environments.

The following example shows the steps for enabling application authenticity on iOS and Android:

1. Modify the `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.

After you have updated your authenticationConfig.xml file, rebuild, and redeploy the .war file.

2. Modify the application-descriptor.xml file of your application.

Remember: In the application-descriptor.xml file, you must also define a security test. For more information, see "Security tests" on page 870.

The procedure is different for Android and for iOS environments.

For iOS

a. Specify the **bundleId** attribute of your application exactly as you defined it in the Apple Developer portal:
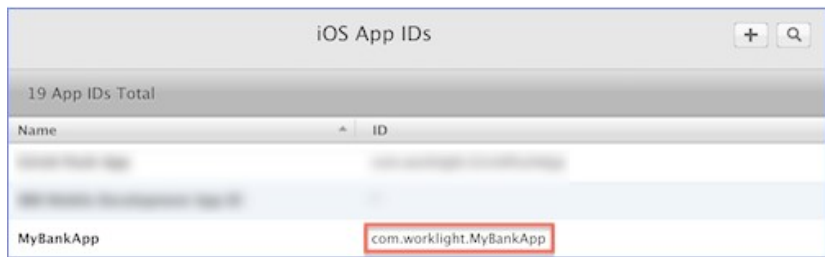
For example:

```
<iphone bundleId="com.worklight.MyBankApp" version="1.0">
    <worklightSettings include="false"/>
    <security>
        <encryptWebResources enabled="false"/>
        <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, j
    </security>
</iphone>
```

b. Add the **applicationId** attribute to the iPhone or iPad element in the application-descriptor.xml file. The **applicationId** value must match the value of the **application id** property, which you can find in the worklight.plist file. For example:

```
<iphone bundleId="com.worklight.MyBankApp" applicationId="MyBankApp" version="1.0">
```

Note: If you decide to change the value of the application ID, ensure that you change it both in the application-descriptor.xml file and in the worklight.plist file.

c. In addition, for a native iOS app, in your XCode project, under **Build Settings** > **Linking** > **Other Linker Flags**, add the -ObjC flag.

For Android

a. 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 that 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. You can find

the development certificate in a keystore under
`{user-home}/.android/debug.keystore`.

You can either extract the public key manually or use a wizard that is provided by Worklight Studio. If you use the wizard, proceed as follows:

1) Right-click your Android environment and select **Extract public signing key**.

2) Specify the location and password of the keystore file and click **Load Keystore**. The default password for `debug.keystore` is `android`.

3) Select key alias and click **Next**.
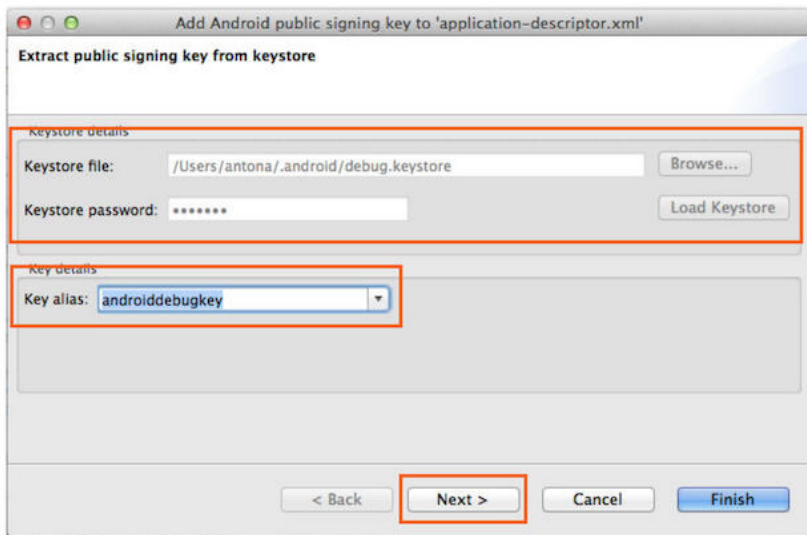
The public key is displayed on a window:
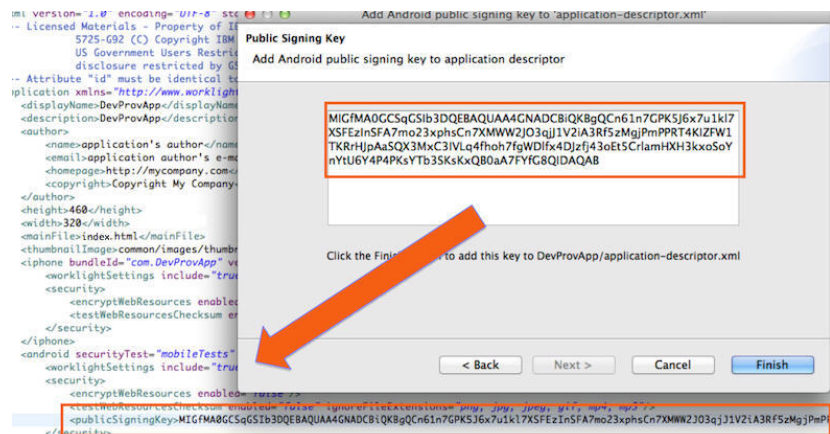
When you click **Finish**, the public key is automatically pasted into the relevant section of `application-descriptor.xml`.

b. Add the application package name to the `application-descriptor.xml` file in either of the following ways:

- In the Application Descriptor editor (accessible from the design view), in the **Application package name** field, add the value of the `package` attribute of the `<manifest>` element in the `AndroidManifest.xml` file.
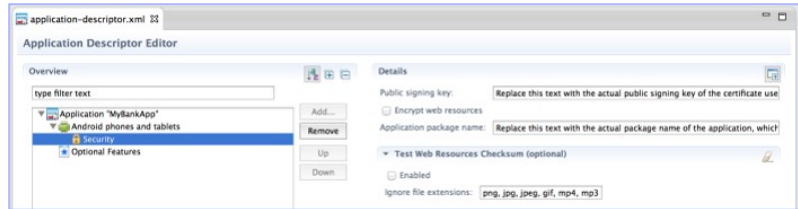
- Use your preferred text editor to edit the application-descriptor.xml file directly. For example:

```
<android securityTest="customTests" version="1.0">
    <worklightSettings include="false"/>
    <security>
        <encryptWebResources enabled="false"/>
        <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg
        <publicSigningKey>my-very-long-public-key</publicSigningKey>
        <packageName>com.myPackageName</packageName>
    </security>
</android>
```

> **Note:** If you decide to change the application package name, ensure that you change it both in the `application-descriptor.xml` file and in the `AndroidManifest.xml` file.

3. After you have updated the required elements, rebuild and redeploy your application to Worklight Server.

Controlling application authenticity from Worklight Console

Through the Worklight Console , you can enable or disable the application authenticity realm at run time. This feature is useful for development and QA environments. You can set one of the following modes:

- **Enabled, blocking**: The application authenticity check is enabled. If the application fails the check, it is not served by Worklight Server.
- **Enabled, serving**: The application authenticity check is enabled. If the application fails the check, it is still served by Worklight Server.
- **Disabled**: The application authenticity check is disabled.
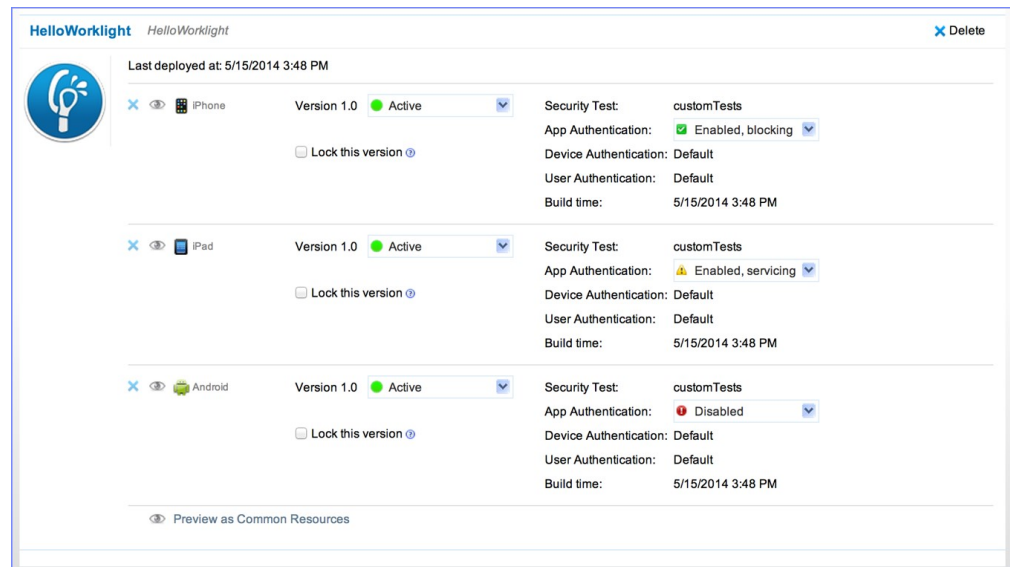
Security tests

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. In web and mobile security tests, you cannot define the sequence in which realms are processed. If you want to define the sequence, you must write your own custom security test and use the **step** property.

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 Foundation checks whether the client is already authenticated according to all realms of the security test. If the client is not yet authenticated, IBM Worklight Foundation 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, automatic checking for Direct Updates every session, 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. Only tests that are included are tested.

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 by 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.

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 1447.

Anti-cross site request forgery (anti-XSRF) realm

The `wl_antiXSRFRealm` protects against cross-site request forgery attacks.

In a cross-site request forgery attack, unauthorized commands are transmitted from a web browser that is trusted by the targeted web site. To protect against this, IBM Worklight Foundation provides an anti-cross site request forgery realm, `wl_antiXSRFRealm`. This realm is enabled by default in the webSecurityTest and the mobileSecurityTest.

The anti-XSRF realm is relevant only for web environments, when the application runs in a browser. It is not relevant for installed mobile applications. Also, the anti-XSRF realm does not protect against session hijacking.

The anti-XSRF technique is based on the same-origin constraint policy, which requires that after an initial request, all subsequent requests come from the same source as the initial one. A script that is loaded from a different origin is assumed to be an attacker script.

When a new session is initiated, the first request to Worklight Server receives an HTTP 401 ("Unauthorized") response that contains the `WL-Instance_Id` token. The MobileFirst security 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, and access to resources is denied.

The server-side realm implementation ensures that each incoming request has the correct value in the `WL-Instance_Id` header. If the header is missing or has an incorrect value, the realm again returns a `401` response with the challenge that contains the correct value for `WL-Instance_Id`. However, due to the same-origin constraint policy, the targeted web site does not allow the attacking web site to read the challenge.

The server returns a challenge and does not destroy the session in the case of a missing or incorrect token because this situation can be a result of a legitimate use case. For example, if a session is timed-out on the server side, the client might send a request with an expired token. Or, a session race condition might occur in which the client sends two or more requests simultaneously when the session is not established or is timed out. A legitimate client should be able to recover from these situations automatically, so the server sends the same challenge in the case of failure.

For more information, see Cross-site request forgery.

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.

Mobile device authentication

You can require mobile devices to authenticate themselves. Device identity is used in several places within IBM Worklight Foundation. 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 Foundation 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:
  - 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.
- On Windows Phone:
  - The publisher host ID is used as a unique device ID. The host ID is unique per device and per publisher, which means that no two publishers will receive the same value for the same device.
- On Windows 8:
  - The MAC address is used as a unique device ID.

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 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 Foundation. 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 1424.

For more information about mobile device provisioning, see the module *Device provisioning concepts*, under category *9. Authentication and security*, in Chapter 4, "Tutorials and samples," on page 35.

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 a Worklight application first runs on a mobile device, it creates a pair of PKI-based keys. It then uses the keys to sign the public characteristics of the device and application, and sends them to the Worklight Server for authentication purposes.

A key pair alone is not sufficient to sign these public characteristics because any app can create a key pair. In order for a key pair to be trusted, it must be signed by an external trusted authority to create a certificate. The process of obtaining such a certificate is called *provisioning*.

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 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: a Worklight CA Certificate, which is embedded into the platform.

Whether it is obtained by an auto-provisioning or custom provisioning process, the certificate is stored by the client app on the device, and used for signing the payload sent to the Worklight Server. The Worklight Server validates the client certificate, regardless of how it is obtained.

The server sends a request for ID, which the client responds to with a certificate-signed payload. If the client does not have the certificate, then a request is sent to the Worklight Server automatically to get a certificate, and after that is done, the client automatically sends the signed payload.

After the server sends the ok response, the original request is sent automatically.

Granularity of provisioning

The key pair that is used to sign the device and app properties can represent a single application, a group of applications, or an entire device. For example:

Single application
A company's provisioning process requires separate activation for each application that is installed on the device. In this case, the application is the provisionable entity, and each application must generate its own key pair.

Group of applications
A company develops different groups of applications to employees in different geographical regions. If the activation is required per region, the key pair would represent the group of applications that belong to that region. All applications from the same group use the same key pair for their signatures.

Entire device
In this case, the key pair represents the whole device. All the applications from the same vendor that are installed on that device use the same key pair.

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 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 143. Predefined realms: properties of the **test realm** *element.*

Realm reference	Login module reference	Description
`wl_anonymousUserRealm`	`WeakDummy`	This realm is the default user realm. As having a user identity is mandatory for a user to use IBM Worklight Foundation properly, use this realm if you do not require any special identification of users. This realm gives the user a random unique user ID to be used for various features in the server, such as reports and audit, identification of access to backend systems, and push notification. This realm is transparent, that is, it does not require any user interaction.
`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_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 Foundation, and is supported by iOS and Android environments only.

Table 143. Predefined realms: properties of the **test realm** *element. (continued)*

Realm reference	Login module reference	Description
`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_deviceNoProvisioningRealm`	`WLDeviceNoProvisioningLoginModule`	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_directUpdateRealm`	`WLDirectUpdateNullLoginModule`	This realm is used to enable the direct update feature. The direct update feature allows for updating of application web resources (not native code) on client devices without the need for users to explicitly download and install the new version. This realm is useful when a fix or an enhancement is done to the web resources of the application and you do not want to start a full release cycle for it. It can be configured to test for updates once per session, per each request, or disabled. For more information about direct update, see "Configuring and customizing direct update" on page 599.

wl_remote_DisableRealm	WLRemoteDisableNullLoginModule	This realm is used to block applications with specific application environments or versions from accessing resources on the server, or to notify clients with some mandatory message that is related to the server. This realm is typically used when a new application version is released and you no longer want the applications with the older versions to connect to the server. In this case, for example, you want to give directions to the clients on how to obtain the new version of the application with a link to its market download page. Another typical use of this realm is when you find a problem with an application security and you want to immediately block access from this application to sensitive data until the problem is fixed. You can configure the contents of the block or notification message and give a link to more information or the new version. For more information about remote disable, see "Remotely disabling application connectivity" on page 1249.

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 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 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 264.

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 144. 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 145. 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.

Implementing basic authenticators

You can implement basic authentication in mobile applications.

About this task

The basic authenticator implements basic HTTP authentication. Basic authentication is an industry-standard method that is used to collect user name and password information.

In accordance with standard basic authentication, Worklight Server sends an HTTP Not Authorized (401) response to the client, with the header: `WWW-Authenticate: Basic realm="realmName"`. When Worklight Server receives the response from the client, it extracts the base64-encoded credentials from the Authorization header of the request and decodes them. A login module validates the credentials that have been received.

Note: You can use basic authentication for web applications only, not for mobile applications.

The fully qualified Java class name for the basic authenticator is:

`com.worklight.core.auth.ext.BasicAuthenticator`

Parameters

The basic authenticator has the following parameter:

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.

Flow

The following diagram illustrates the flow in the basic authentication process:

Figure 109. Basic authentication processBasic authentication process

Procedure

1. Configure the `authenticationConfig.xml` file. For more information, see "The authentication configuration file" on page 877.
2. Code the server side.

   Note: If you want to protect an adapter procedure with basic authentication, you must declare it in the adapter XML file. See the example in this page.
3. Associate the basic authenticator with a login module. Worklight Studio provides several predefined login modules. For an example, see Non-validating login module.
4. Code the client side, if necessary.

Example

The following example demonstrates how to implement a simple basic authentication mechanism. An adapter procedure is protected by a basic authenticator, and when the user attempts to invoke the procedure from the application, the browser displays a login dialog and the authentication process starts.
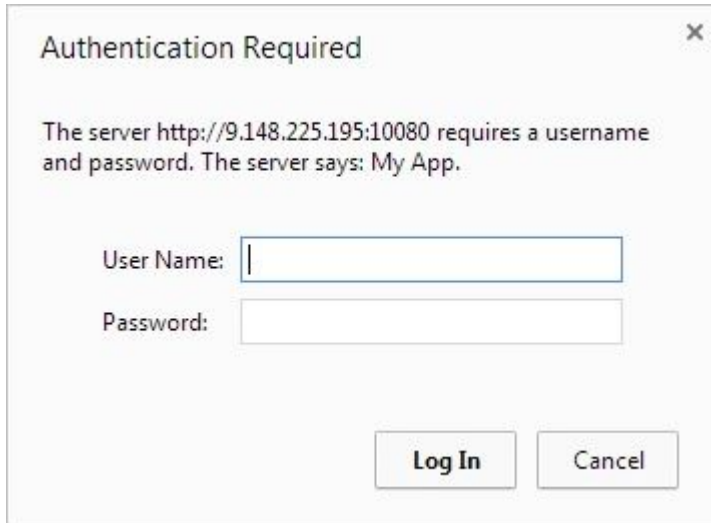
Figure 110. Login dialog for authentication

Configuration of the `authenticationConfig.xml` file

```xml
<securityTests>
  <customSecurityTest name="DummyAdapter-securityTest">
    <test isInternalUserID="true" realm="MyAppRealm"/>
  </customSecurityTest>
</securityTests>

<realms>
  <realm name="MyAppRealm" loginModule="StrongDummy">
    <className>com.worklight.core.auth.ext.BasicAuthenticator</className>
      <parameter name="basic-realm-name" value="My App"/>
  </realm>
</realms>

<loginModules>
  <loginModule name="StrongDummy">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule
    </className>
  </loginModule>
</loginModules>
```

Note:

The realm uses the StrongDummy login module, which is implemented by the class `NonValidatingLoginModule` (see Non-validating login module). "Non-validating" means that the user credentials are not checked against any list of user names and passwords. In other words: any combination of user name and password is valid.

Coding the server side

1. Create a Worklight adapter.
2. Add a procedure and protect it with the custom security test that you created earlier. This procedure's implementation can return some hard-coded value, for example:

   ```xml
   <procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/>
   ```

Coding the client side

1. Create a Worklight application.
2. Write a call to the adapter procedure that you added on the server side, for example:

```
                var invocationData = {
                  adapter: <adapterName>,
                  procedure: "getSecretData",
                  parameters: []
                };

                WL.Client.invokeProcedure(invocationData, {
                  onSuccess : successCallback,
                  onFailure : failCallback
                });
```

Implementing form-based authenticators

You can authenticate users of mobile applications by using a login form.

About this task

In form-based authentication, if an application tries to access a protected resource,
the server returns the HTML code of a login form. Even though a form of this kind
is most suited to desktop and web environments (where you display the returned
login form), you can also use form-based authentication in mobile applications.

The fully qualified Java class name of the form-based authenticator is:
`com.worklight.core.auth.ext.FormBasedAuthenticator`.

This authenticator type presents a login form to the user. The login form must
contain `j_username` and `j_password` fields, the `j_security_check` submit action, and
the `POST` submit method.

A login module validates the credentials that are provided. If the login fails, the
user is redirected to an error page.

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 you create 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}` in the login page template.
`auth-redirect`	Path to a user-defined login page (`html/jsp`) relative to the web application context. IBM Worklight Foundation redirects to the page when the user credentials are needed.

> Both the **`login-page`** and **`auth-redirect`** parameter are optional, but if you
> decide to use them, use either one or the other. You cannot use them
> together. You can also use neither. In this case, IBM Worklight Foundation
> uses its default login page template.

Flow

> The following diagram illustrates the flow in the form-based authentication
> process:
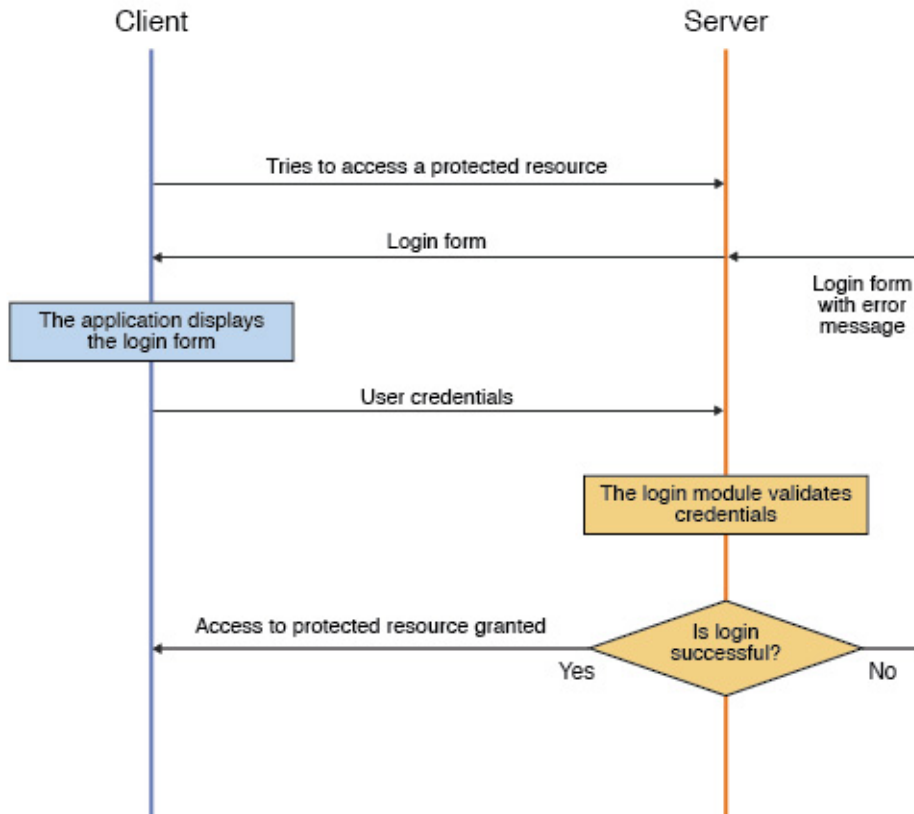
Figure 111. Form-based authentication process

Procedure

1. Configure the `authenticationConfig.xml` file. For more information, see "The authentication configuration file" on page 877.

2. Code the server side. To work, the form-based authenticator must be associated with a login module. Worklight Studio provides several predefined login modules. For an example, see Non-validating login module.

   Note: If you want to protect an adapter procedure with form-based authentication, you must declare it in the adapter XML file. See the example in this page.

3. Code the client side.

   You must declare a challenge handler in the application to handle challenges from the form-based configured realm. The following sample shows one way to implement a challenge handler class:

   ```
   var sampleAppRealmChallengeHandler = WL.Client.createChallengeHandler("SampleAppRealm");
   ```

   The challenge handler must implement the following functions:

   - `isCustomResponse`: this function is called each time that a response is received from the server. It is used to detect whether the response contains data that is related to this challenge handler. It must return either `true` or `false`. Here is a simple example:

   ```
   sampleAppRealmChallengeHandler.isCustomResponse = function(response) {
     return false;
   };
   ```

- **handleChallenge**: this function is used to perform required actions, such as hide application screen and show login screen. `handleChallenge` is called by the framework, if `isCustomResponse` returns `true`. Here is a simple implementation, as an example:

```
sampleAppRealmChallengeHandler.handleChallenge = function(response) {
};
```

The challenge handler can also optionally implement the following, additional functions:

- **submitLoginForm**: this function sends the collected credentials to a specific URL. You can also specify request parameters, headers, and callback.
- **submitSuccess**: this function notifies the Worklight framework that the authentication finished successfully. The Worklight framework then automatically issues the original request that triggered the authentication.
- **submitFailure**: this function notifies the Worklight framework that the authentication process failed to finish. The Worklight framework then disposes of the original request that triggered the authentication.

Example

The following example demonstrates how to implement a simple form-based authentication mechanism that is based on a user name and a password. In the example, an adapter procedure is protected by a form-based authenticator, and when the user attempts to call the procedure from the application, the login form is displayed and the authentication process starts.

Configuration of the `authenticationConfig.xml` file

```
<securityTests>
  <customSecurityTest name="DummyAdapter-securityTest">
    <test isInternalUserID="true" realm="SampleAppRealm"/>
  </customSecurityTest>
</securityTests>

<realms>
  <realm name="SampleAppRealm" loginModule="StrongDummy">
    <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
      <parameter name="login-page" value="login.html"/>
  </realm>
</realms>

<loginModules>
  <loginModule name="StrongDummy">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
  </loginModule>
</loginModules>
```

Coding the server side

Perform the following steps:

1. Create a Worklight adapter.
2. Add a procedure and protect it with the custom security test that you created earlier. The implementation can return some hard-coded value, for example:

   ```
   <procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/>
   ```

Coding the client side

Perform the following steps:

1. Create a Worklight application.

2. Create a challenge handler in the application, to handle challenges from the `SampleAppRealm` realm, for example:

```
var sampleAppRealmChallengeHandler = WL.Client.createChallengeHandler("SampleAppRealm"
```

3. Implement the mandatory `isCustomResponse` and `handleChallenge` functions (and other, optional functions) of the challenge handler, as described previously.

What to do next

For a more extensive example of implementing form-based authentication, see the module *Form-based authentication*, under category 8, *Authentication and security*, in Chapter 4, "Tutorials and samples," on page 35.

Implementing custom authenticators

You can use default Worklight login modules and authenticators, or customize your own.

About this task

You can write custom login modules and authenticators when those that IBM Worklight Foundation supplies do not match your requirements.

Procedure

1. Configure the `authenticationConfig.xml` file.

   For more information, see "The authentication configuration file" on page 877.

2. Code the server side.

   You create custom login modules and authenticators as instances of Java™ classes, which you must place in the `server/java` folder of the project. They are server-side entities and they are packed inside the WAR file of the project. The authenticator, login module, and user identity instances are stored in a session scope, so that they exist while the session is active.

   Authenticator interface and methods

   Your custom authenticator class must implement the `com.worklight.server.auth.api.WorkLightAuthenticator` interface. The custom authenticator must implement the following methods:

   - `init`: This method is called when the authenticator instance is created. It receives the options that are specified in the realm definition in the `authenticationConfig.xml` file.
   - `processRequest`: This method is called for each request from an unauthenticated session. The method must return an `AuthenticationResult` status. While the request is processed, the method might retrieve data from the request and write data to the response.

     The `AuthenticationResult` status can return the following values:

     – `SUCCESS`: The credentials were successfully collected and the login module can now validate them.
     – `CLIENT_INTERACTION_REQUIRED`: The client must still supply authentication data.
     – `REQUEST_NOT_RECOGNIZED`: The authenticator is not handled.
   - `processAuthenticationFailure`: This method is called if the login module returns a failure for the validation of credentials.

- processRequestAlreadyAuthenticated: This method is called for each request from a session that has already been authenticated. It returns an AuthenticationResult value for authenticated requests.
- getAuthenticationData: Login modules use this method to retrieve the credentials that are collected by an authenticator.
- changeResponseOnSuccess: This method is called after the login module successfully validates credentials. Use this method to notify a client application of the success of the authentication, for example to modify the response before it is returned to the client. This method must return true if the response was modified orfalse otherwise.
- clone: This method creates a deep copy of the object members.

Login module interface and methods

Your custom login module class must implement the com.worklight.server.auth.api.WorkLightAuthLoginModule interface. The login module must implement the following methods:

- init: This method is called when the login module instance is created. This method receives the options that are specified in the login module definition of the authenticationConfig.xml file.
- login: This method is called after the authenticator returns SUCCESS status. It receives an authenticationData object from the authenticator and validates the credentials that are collected by the authenticator. If the credentials are valid, the method returns true. If the credential validation fails, the method returns false or raises a runtime exception. In this case, the exception string that is returned to the authenticator as an **errorMessage** parameter.
- createIdentity: This method is called after the credentials are successfully validated. The method creates and returns a UserIdentity object, which contains information about the authenticated user, such as unique user name, display name, Java security roles, and custom user attributes.
- logout: Use this method to clean up cached data and class members after the user logs out.
- abort: Use this method to clean up cached data and class members after the user stops the authentication flow.
- clone: This method creates a deep copy of the object members.

3. Code the client side.

You must declare a challenge handler in the application to handle challenges from the custom authenticator realm. The following sample shows one way to implement a challenge handler class:

```
var myChallengeHandler = WL.Client.createChallengeHandler("CustomAuthenticatorRealm");
```

The challenge handler must implement the following methods:

- isCustomResponse: This method is called each time that a response is received from the server. It detects whether the response contains data that is related to this challenge handler. It must return true or false. Here is a simple example:

```
sampleAppRealmChallengeHandler.isCustomResponse = method(response) {
  return false;
};
```

- handleChallenge: Use this method for such actions as hide application screen and show login screen. If the isCustomResponse method returns true, thehandleChallenge method is called by the framework. Here is a simple implementation, as an example:

```
sampleAppRealmChallengeHandler.handleChallenge = method(response) {
};
```

Optionally, the challenge handler can also implement the following methods:

- `submitLoginForm`: This method sends the collected credentials to a specific URL. You can also specify request parameters, headers, and callback.
- `submitSuccess`: This method notifies the Worklight framework that the authentication finished successfully. The Worklight framework then automatically issues the original request that triggered the authentication.
- `submitFailure`: This method notifies the Worklight framework that the authentication process failed. The Worklight framework then disposes of the original request that triggered the authentication.

Example

The following example shows how to implement a custom authenticator and login module. In the example, an adapter procedure is protected by a custom authenticator. When the user attempts to call the procedure from the application, the application requests the user's credentials and the authentication process starts.

Configuration of the `authenticationConfig.xml` file
```
<securityTests>
  <customSecurityTest name="DummyAdapter-securityTest">
      <test isInternalUserID="true" realm="CustomAuthenticatorRealm"/>
   </customSecurityTest>
</securityTests>

<realms>
  <realm name="CustomAuthenticatorRealm" loginModule="CustomLoginModule">
      <className>com.mypackage.MyCustomAuthenticator</className>
  </realm>
</realms>

<loginModules>
  <loginModule name="CustomLoginModule">
      <className>com.mypackage.MyCustomLoginModule</className>
  </loginModule>
</loginModules>
```

Coding the server side

Code the following elements on the server side: adapter, authenticator, and login module.

- Adapter:
  1. Create a Worklight adapter.
  2. Add a procedure and protect it with the custom security test that you created earlier. The implementation can return some hardcoded value. For example:
     ```
     <procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/>
     ```
- Authenticator:
  1. Create a `MyCustomAuthenticator.java` class in the `server/java/com/mypackage` folder. This class must implement the `com.worklight.server.auth.api.WorkLightAuthenticator` interface, as follows:
     ```
     public class MyCustomAuthenticator implements WorkLightAuthenticator{}
     ```
  2. Implement the mandatory methods of the class.
     - `processRequest`: This method retrieves the user name and password credentials that are passed as request parameters. Check the credentials for basic validity, create an `authenticationData`

object, and return SUCCESS. If a problem occurs with the received
credentials, add an `errorMessage` to the response and return the
`CLIENT_INTERACTION_REQUIRED` status message. If the request does
not contain authentication data, add the `authStatus:required`
property to the response and again, return a
`CLIENT_INTERACTION_REQUIRED` status message.

```
public AuthenticationResult processRequest(HttpServletRequest request, HttpServletRe
  boolean isAccessToProtectedResource) throws IOException, ServletException {
  if (request.getRequestURI().contains("my_custom_auth_request_url")){
    String username = request.getParameter("username");
    String password = request.getParameter("password");

    if (null != username && null != password && username.length() > 0 && password.le
      authenticationData = new HashMap<String, Object>();
      authenticationData.put("username", username);
      authenticationData.put("password", password);
      return AuthenticationResult.createFrom(AuthenticationStatus.SUCCESS);
    } else {
      response.setContentType("application/json; charset=UTF-8");
      response.setHeader("Cache-Control", "no-cache, must-revalidate");
      response.getWriter().print("{\"authStatus\":\"required\", \"errorMessage\":\"F
      return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION
    }
  }
  if (!isAccessToProtectedResource)
      return AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNI
  response.setContentType("application/json; charset=UTF-8");
  response.setHeader("Cache-Control", "no-cache, must-revalidate");
  response.getWriter().print("{\"authStatus\":\"required\"}");
  return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQ
}
```

– `processAuthenticationFailure`: This method writes an error
message to a response body and returns the
`CLIENT_INTERACTION_REQUIRED` status message.

```
public AuthenticationResult processAuthenticationFailure(HttpServletRequest
  request, HttpServletResponse response, String errorMessage) throws IOException, S
  response.setContentType("application/json; charset=UTF-8");
  response.setHeader("Cache-Control", "no-cache, must-revalidate");
  response.getWriter().print("{\"authStatus\":\"required\", \"errorMessage\":\"" + e
  return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQ
}
```

- Login module:
  1. Create a `MyCustomLoginModule.java` class in the `server/java/com/`
     `mypackage` folder. This class must implement the
     `com.worklight.server.auth.api.WorkLightAuthLoginModule` interface.

     ```
     public class MyCustomLoginModule implements WorkLightAuthLoginModule{}
     ```

  2. Implement the mandatory methods of the class.

     – `login`: This method retrieves the user name and password
     credentials that the authenticator stored previously. In this
     example, the login module validates the credentials against
     hardcoded values. You can implement your own validation rules.
     If the credentials are valid, the login method returns `true`. For
     example:

     ```
     public boolean login(Map<String> authenticationData) {
       USERNAME =(String) authenticationData.get("username");
       PASSWORD = (String) authenticationData.get("password");
       if (USERNAME.equals("wluser") && PASSWORD.equals("12345"))
       return true;
       else    throw new RuntimeException("Invalid credentials"); }
     </String>
     ```

- – createIdentity: This method is called when the `login` method returns `true`. It is used to create a `UserIdentity` object. In that object, you can store your own custom attributes and use them later in Java or adapter code. The `UserIdentity` object contains user information. Its constructor is as follows:

```
public UserIdentity(String loginModule, String name, String displayName, Set<Stri
```

Here is an example of how to implement this method:

```
public UserIdentity createIdentity(String loginModule) {
  HashMap<String, Object> customAttributes = new HashMap<String, Object>();
  customAttributes.put("AuthenticationDate", new Date());

  UserIdentity identity = new UserIdentity(loginModule, USERNAME, null, null, cus
  return identity;
}
```

Coding the client side

Follow these steps:

1. Create a Worklight application.
2. Create a challenge handler in the application to handle challenges from the custom authenticator realm. For example:

   ```
   var myAppRealmChallengeHandler =
   WL.Client.createChallengeHandler ("CustomAuthenticatorRealm");
   ```

3. Implement the mandatory `isCustomResponse` and `isCustomResponse` methods, and optional methods of the challenge handler, as described in Step 3.

What to do next

For a more extensive example of implementing custom authentication and login, see module *Custom Authenticator and Login Module in hybrid applications* in Chapter 4, "Tutorials and samples," on page 35.

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>
```

Implementing adapter-based authenticators

You can authenticate users of mobile applications by using an adapter-based authenticator.

About this task

Adapter-based authentication allows you to develop custom authentication logic by using a JavaScript function within a Worklight® adapter.

Flow

> Adapter-based authentication is very flexible and customizable. The following diagram illustrates one possible implementation. A description follows.
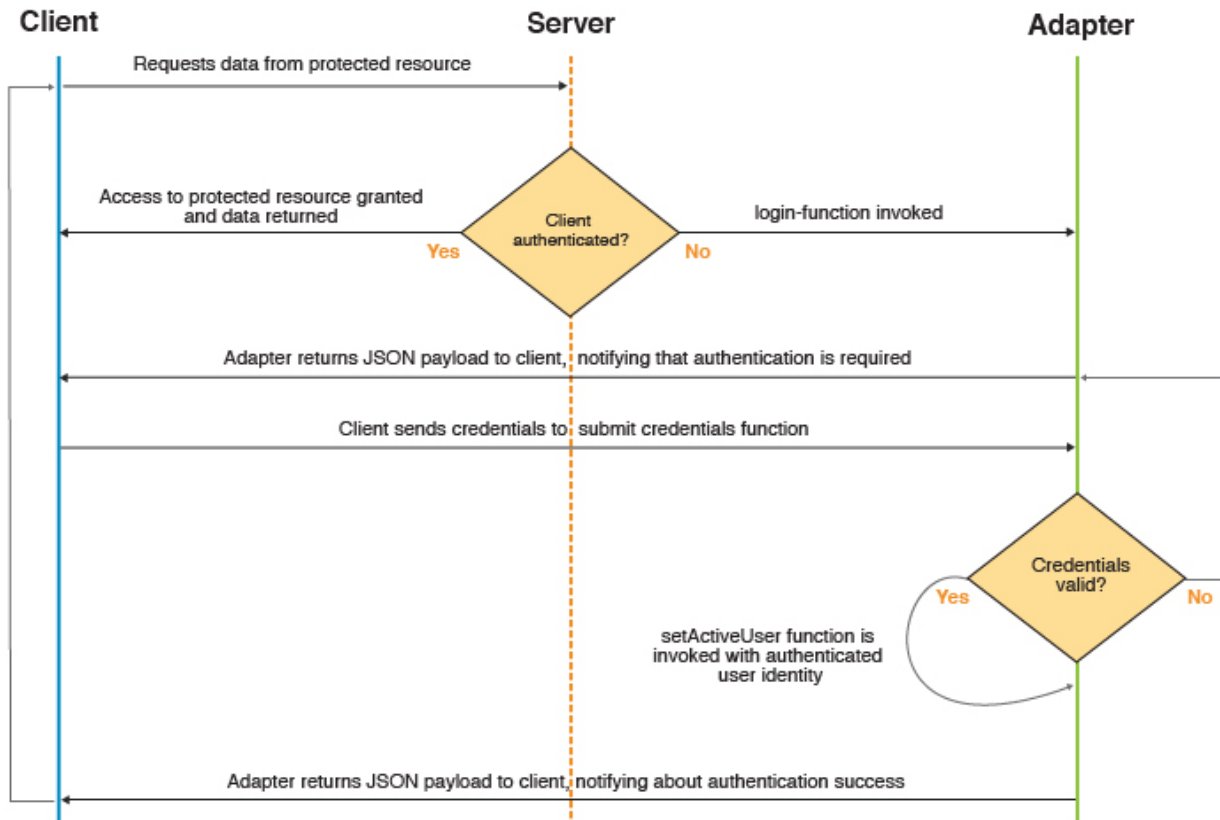
Figure 112. Adapter-based authentication process

1. The client makes a request to the resource that is protected by adapter authentication.
2. Worklight Server checks whether the client is already authenticated.
   a. If it is, the requested data is returned.
   b. Otherwise, authentication continues.
3. The adapter procedure that is defined in `authenticationConfig.xml` as a login-function is invoked.
4. The login-function procedure is used to return a custom JSON payload to the client.
5. The client processes the custom JSON payload and sends its credentials to the adapter procedure used for authentication.
6. The adapter procedure that is used for authentication receives credentials and validates them.
   a. If validation fails, the flow returns to step 4.
   b. Otherwise, authentication continues.
7. The adapter procedure that is used for authentication creates a user identity and returns a success status to the client.
8. The client receives the success status and issues the original request.
9. The flow returns to step 2.

For more information, see "The authentication configuration file" on page 877.

Procedure

1. Configure the `authenticationConfig.xml` file.
   - Add security tests to the `<securityTest>` section of the file. Because the security test that you are using is protecting an adapter procedure, you use the `<customSecurityTest>` parameter.
   - Add authentication realms to the `<realms>` section. For the `className` parameter, use the `com.worklight.integration.auth.AdapterAuthenticator` to indicate that the server-side part of the authenticator is defined in the adapter. Define two parameter-value pairs for login and logout:
     – `login-function`: whenever the Worklight authentication framework detects an attempt to access a protected resource, the login-function is called automatically.
     – `logout-function`: when logout is detected (explicit or session timeout), the logout-function is called automatically.

     In both cases, the value syntax is `adapterName.functionName`.
   - Add a login module to the `<loginModules>` section. All of the validation logic that is usually done in a login module is performed in the adapter's JavaScript code and you need no further validation. For that reason, adapter-based authentication must be used with a `NonValidatingLoginModule` only. No additional validation is performed by the Worklight platform, and the developer takes responsibility for the validation of credentials within the adapter. For more information, see Non-validating login module.
2. Code the server side. The adapter authenticator fully qualified Java™ class name is `com.worklight.integration.auth.AdapterAuthenticator`, and it has two parameters:
   a. `login-function` (mandatory)
   b. `logout-function` (optional)

   Both parameters specify adapter function names. The syntax is: `adapter-name.function-name`, for example, `myAuthAdapter.onAuthRequired`. You need to implement the `login-function` and `logout-function` (`AuthAdapter.onAuthRequired` and `AuthAdapter.onLogout` in the example below) in your `adapter.js` source file.

   Note:
   - 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.
   - In contrast, the function that receives credentials is directly called by a client. Therefore, you must expose the function in the adapter XML file. When the challenge handler invokes the submit call, the handler is responsible for handling all the possible responses. In particular, if the submit call returns a challenge, the challenge is passed to the invocation callback, and is not processed by the security framework. To prevent a situation in which the invocation callback cannot handle the challenge, disable the authentication requirement for the submit procedure by using the wl_unprotected security test.
   - Alternatively, you can define a more sophisticated security test for this function. Just make sure the security context on the client side is sufficient to answer the challenge. One way to do this is to enrich the client security context by calling `WL.Client.connect()` before invoking an adapter.

- If your Worklight Server runs on WebSphere® Application Server, version 7 (any release) or releases of WebSphere Application Server Liberty 8.5 prior to Fix Pack 2, the application server's Web container custom flag

  `com.ibm.ws.webcontainer.suppressLoggingServiceRuntimeExcep`

  must be set to true. The default is false. If this flag is not changed, then the adapter will fail to authenticate and an exception will occur. For more information, see APAR PM74090 or APAR PM79934 for WebSphere Application Server.

In addition to implementing login-function and logout-function, you also need to implement an adapter function that receives 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 either when 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 to notify it about the required authentication. The login-function receives original request headers that are converted to JSON as a first function argument so that they can be used to decide on the kind of authentication that is needed, for example. Then it is the login-function that returns the response to the client, instead of the original function.

- **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.

- **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. If not, it is advised to programatically limit the number of login trials in your application.

3. Code the client side.

   a. Create a Worklight application, with an element for displaying the application content and an element for authentication. For example, when authentication is required, the application hides the applicative element and shows the authentication element. When authentication is complete, it does the opposite.

   b. Create a challenge handler, by using the `WL.Client.createChallengeHandler` method to create a challenge handler object. You must *implement* the following mandatory methods: `isCustomResponse`, `handleChallenge`. In addition, the following mandatory methods are available in every challenge handler that you must *use*: `submitAdapterAuthentication`, `submitSuccess`, `submitFailure`.

Note:
You must attach each of these mandatory challenge handler functions to its object. For example: `myChallengeHandler.submitSuccess`.

Example

The following example demonstrates how to implement an adapter-based authentication mechanism that relies on a user name and a password.

Configuration of the `authenticationConfig.xml` file

```
<securityTests>
  <customSecurityTest name="SingleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="SingleStepAuthRealm"/>
  </customSecurityTest>
</securityTests>

<realms>
  <realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
      <parameter name="login-function"
        value="SingleStepAuthAdapter.onAuthRequired"/>
      <parameter name="logout-function"
        value="SingleStepAuthAdapter.onLogout"/>
  </realm>
</realms>

<loginModules>
  <loginModule name="AuthLoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule
    </className>
  </loginModule>
</loginModules>
```

Code the server side authentication

Perform the following steps:

1. Create an adapter that takes care of the authentication process. In this example, it is `SingleStepAuthAdapter`.

2. `SingleStepAuthAdapter` could include the following two procedures, for example:

   ```
   <procedure name="submitAuthentication" securityTest="wl_unprotected"/>
   <procedure name="getSecretData" securityTest="SingleStepAuthAdapter-securityTest"/>
   ```

   - The `submitAuthentication` procedure takes care of the authentication process and authentication is not required to call it.
   - The `getSecretData` procedure is available to authenticated users only.

3. Define the `onAuthRequired` function:

   ```
   function onAuthRequired(headers, errorMessage) {
     errorMessage = errorMessage ? errorMessage : null;

     return {
       authRequired: true,
       errorMessage: errorMessage
     };
   }
   ```

   - This function receives the response headers and an optional `errorMessage` parameter. The object that is returned by this function is sent to the client application. The `authRequired:true` and `errorMessage:errorMessage` pairs define a custom challenge object that is sent to the application.

- The `authRequired:true` property is used in a challenge handler to detect that the server is requesting authentication.
- Whenever the Worklight framework detects an unauthenticated attempt to access a protected resource, the `onAuthRequired` function is called, as you defined in the `authenticationConfig.xml` file.

4. Define the `submitAuthentication` function. The function is called by the client app to validate the user name and password.

```
/* In this sample, the credentials are validated against some
 * hardcoded values, but any other validation mode is valid,
 * for example by using SQL or web services. */
if (username==="worklight" && password === "worklight"){

/* If the validation passed successfully, the WL.Server.setActiveUser method
 * is called to create an authenticated session for the SingleStepAuthRealm,
 * with user data stored in a userIdentity object. You can add your own custom
 * properties to the user identity attributes. */
var userIdentity = {
  userId: username,
  displayName: username,
  attributes: {
    foo: "bar"
  }
};

WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

 /* An object is sent to the application, stating that the authentication
  * screen is no longer required. */
 return {
   authRequired: false
 };
}

/* If the credentials validation fails, an object that is built
 *   by the onAuthRequired function is returned to the application
 *   with a suitable error message. */
 return onAuthRequired(null, "Invalid login credentials");
}
```

5. Define the `getSecretData` function. For the purposes of demonstration, at the conclusion of successful authentication, you could return a hard-coded value:

```
function getSecretData() {
  return {
    secretData: "Very very secret data"
  };
}
```

6. Define the `onLogout` function, to be called automatically on logout. It can perform a cleanup, for example.

```
function onLogout(){
  WL.Server.setActiveUser("SingleStepAuthRealm", null);
  WL.Logger.debug("Logged out");
}
```

Code the client side authentication

Perform the following steps:

1. Create a Worklight application.
2. You might create some HTML code, for example, to display application content only after authentication is complete.

3. Create the challenge handler. Use the
   `WL.Client.createChallengeHandler` method to create a challenge
   handler object; supply a realm name as a parameter. For example:

```
var singleStepAuthRealmChallengeHandler =
  WL.Client.createChallengeHandler("SingleStepAuthRealm");

/* The isCustomResponse function of the challenge handler
 * is called each time a response is received from the server.
 * That function is used to detect whether the response contains
 * data that is related to this challenge handler. The function returns true or false.
 */

singleStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
  if (!response||!response.responseJSON||response.responseText === null) {
    return false;
  }
  if (typeof(response.responseJSON.authRequired) !== 'undefined'){
   return true;
  } else {
   return false;
  }
};
```

4. Define a `handleChallenge` function. That function behaves differently
   according to the result of the `authRequired` function in the previous
   step.

```
/* If the isCustomResponse function returns true, the
 * framework calls the handleChallenge function. This function
 * is used to perform required actions, such as to hide the
 * application screen or show the login screen. */
singleStepAuthRealmChallengeHandler.handleChallenge =
  function(response){
  var authRequired = response.responseJSON.authRequired;

  if (authRequired == true){
  $("#AppDiv").hide();
  $("#AuthDiv").show();
  $("#AuthPassword").empty();
  $("#AuthInfo").empty();

  if (response.responseJSON.errorMessage)
      $("#AuthInfo").html(response.responseJSON.errorMessage);

  } else if (authRequired == false){
    $("#AppDiv").show();
    $("#AuthDiv").hide();
    singleStepAuthRealmChallengeHandler.submitSuccess();
  }
};
$("#authCancelButton").click(function(){
  singleStepAuthRealmChallengeHandler.submitFailure();
});
```

The code in this step demonstrates two of three additional challenge
handler functions that you need to use:

- The `submitSuccess` function notifies the Worklight framework that
  the authentication process completed successfully. The Worklight
  framework then automatically issues the original request that
  triggered authentication.
- The `submitFailure` function notifies the Worklight framework that
  the authentication process completed with failure. The Worklight
  framework then disposes of the original request that triggered
  authentication.

5. The third challenge handler function you must use is
   `submitAdapterAuthentication`. It sends collected credentials to a
   specific adapter procedure. It has the same signature as the
   `WL.Client.invokeProcedure` function. Here is an example:

```
$("#AuthSubmitButton").bind('click', function () {
  var username = $("#AuthUsername").val();
  var password = $("#AuthPassword").val();

  var invocationData = {
    adapter : "SingleStepAuthAdapter",
    procedure : "submitAuthentication",
    parameters : [ username, password ]
  };

  singleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
});
```

What to do next

For a more extensive example of implementing form-based authentication, see the
module *Form-based authentication*, under category 8, *Authentication and security*, in
Chapter 4, "Tutorials and samples," on page 35.

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`.

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. **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.
`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">
<className> com.worklight.core.auth.ext.NonValidatingLoginModule </className>
</loginModule>
```

Single identity login module

The single identity login module is used to grant access to a protected resource 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 protected resource.
`console.password`	Password of the user who can access the protected resource. The password can be encrypted as indicated in "Storing properties in encrypted format" on page 1178.

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`.

Parameter	Description
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`

Parameter	Description	Sample values
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
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.

Value	Description
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.

Configuring device auto provisioning

You can change the default behavior of device auto provisioning with regards to granularity of the provisioning, and pre-required realms for provisioning. You can also change the CA certificate (root certificate) that is used to issue certificates for provisioned devices.

Procedure

- 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.

- To use a CA certificate other than the default Worklight CA certificate, configure the following properties.

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`.

For information about how to specify Worklight configuration properties, see "Configuration of Worklight applications on the server" on page 1171

- To enable multiple applications to share the same certificate, define a **`sharedUserId`** attribute (for Android) or a **`bundleId`** attribute (for iOS) in the application descriptor. For further information about defining these attributes, see "The application descriptor" on page 430.

Configuring and implementing custom device provisioning

Custom device provisioning is an extension of auto device provisioning. The main difference between auto and custom provisioning is that you can perform custom validation of the certificate signing request (CSR) during the provisioning process and custom validation of the certificate during each device authentication process.

The custom device provisioning must be implemented in the JavaScript code of an adapter. The names of the validate-csr and validate-certificate functions must be put in the `authenticationConfig.xml` file as realm and login module parameters:

```
<securityTests>
  <mobileSecurityTest name="CustomDeviceProvisioningSecurityTest">
    <testAppAuthenticity/>
      <testDeviceId provisioningType="custom" realm="CustomDeviceProvisioningRealm"/>
  </mobileSecurityTest>
</securityTests>

<realms>
  <realm name="CustomDeviceProvisioningRealm" loginModule="CustomDeviceProvisioningLoginModule">
    <className>com.worklight.core.auth.ext.DeviceAutoProvisioningAuthenticator</className>
    <parameter name="validate-csr-function" value="ProvisioningAdapter.validateCSR"/>
  </realm>
</realms>

<loginModules>
  <loginModule name="CustomDeviceProvisioningLoginModule">
```

```
            <className>com.worklight.core.auth.ext.DeviceAutoProvisioningLoginModule</className>
            <parameter name="validate-certificate-function" value="ProvisioningAdapter.validateCertifica
        </loginModule>
    </loginModules>
```

The *validate-csr-function* is used to check that the certificate signing request (CSR) sent by the client is complete, and contains the correct information that is needed for the certification of the device. This logic might also validate some properties of CSR against internal or external services / directories if required.

The *validate-certificate-function* is used to verify that the certificate was issued with the right CA. The logic might also verify that the certificate contains all the data about the device that is required for this custom device authentication realm.

For more information about how to implement these functions, see the *Custom Device Provisioning* module under category 9, *Authentication and security*, in Chapter 4, "Tutorials and samples," on page 35.

It is important to understand the concept of mobile device authentication and auto provisioning. For more information about mobile device authentication, see "Mobile device authentication" on page 874.

With custom device provisioning, you can also implement custom variations of:
* The CSR during the initial provisioning flow
* The certificate during every application startup

Server-side and client-side components

You must configure the server and the client for custom device provisioning. The following sections describe the implementations of the server-side and client-side components.

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." For more information
about custom device provisioning, see the module *Custom Authenticator and Login
Module in hybrid applications* under category 9, *Authentication and security*, in
Chapter 4, "Tutorials and samples," on page 35.

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 with IBM Worklight Consumer Edition, and IBM Worklight Enterprise Edition. For more information about application authenticity, see "Worklight application authenticity overview" on page 865.

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.

```
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" on page 907. For more information about custom device provisioning, see the module *Custom device provisioning* under category 8, *Authentication and security*, in Chapter 4, "Tutorials and samples," on page 35.

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, iOS, Windows 8, and Windows Phone 8 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

Single sign-on is a property of a login module. You can enable single sign-on from a mobileSecurityTest element or from a loginModule element of `authenticationConfig.xml`. Enabling single sign-on for a custom security test is done on the loginModule element. Enabling single sign-on for a mobile security test is done on the testUser realm of the mobileSecurityTest element.

Procedure

Take the following points into consideration depending on how you choose to configure device single sign-on:

- 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. If the **sso** attribute is not specified, it is assumed to be set to `false`. 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, where **ssoDeviceLoginModule** is the name of the login module used for the device authentication realm. 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. When configuring SSO on mobileSecurityTest, the platform takes care of this prioritization automatically. 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>
```

- For Windows Phone 8, the following items must be implemented:

  - The Publisher ID specified in the WMAppManifest.xml file must be the same for all applications that participate in the single sign-on.

  - The following line must be added to the WMAppManifest.xml file:

    ```
    <Capability Name='ID_CAP_IDENTITY_DEVICE' />
    ```

- 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 Worklight configuration properties, see "Configuration of Worklight applications on the server" on page 1171.

Results

Device single sign-on implementations are successful if they conform to any of the following valid configurations. Avoid inconsistent states resulting from configurations with built-in conflicts, as described below. Inconsistent states can result in the Worklight project failing to start.

Valid configurations:

- The loginModule element does not specify the **ssoDeviceLoginModule** attribute, and all mobile security tests that use this login module for their user realms have sso="false". In this case, SSO is disabled for all applications that are protected by security tests (mobile or custom) with this login module for a user realm. For example:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
```

```
                <testUser realm="SampleAppRealm" sso="false"/>
            </mobileSecurityTest>
            <mobileSecurityTest name="AnotherFormTest">
                <testDeviceId provisioningType="none"/>
                <testUser realm="SampleAppRealm" sso="false"/>
            </mobileSecurityTest>
    </securityTests>
    <realms>
        <realm name="SampleAppRealm" loginModule="StrongDummy">
            <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
        </realm>
    </realms>
    <loginModules>
        <loginModule name="StrongDummy">
            <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
        </loginModule>
    </loginModules>
```

- The loginModule element does not specify the **ssoDeviceLoginModule**
  attribute, and all mobile security tests that use this login module for
  their user realms have sso="true". In this case, SSO is enabled for all
  applications that are protected by security tests (mobile or custom) with
  this login module for a user realm. For example:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
    <mobileSecurityTest name="AnotherFormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
</securityTests>
<realms>
    <realm name="SampleAppRealm" loginModule="StrongDummy">
        <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
    </realm>
</realms>
<loginModules>
    <loginModule name="StrongDummy">
        <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
    </loginModule>
</loginModules>
```

- The loginModule element specifies the **ssoDeviceLoginModule** attribute,
  and all mobile security tests that use this login module for their user
  realms have sso="true". In this case, SSO is enabled for all applications
  that are protected by security tests (mobile or custom) with this login
  module for a user realm. For example:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
    <mobileSecurityTest name="AnotherFormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
</securityTests>
<realms>
    <realm name="SampleAppRealm" loginModule="StrongDummy">
        <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
    </realm>
</realms>
<loginModules>
```

```
            <loginModule name="StrongDummy" ssoDeviceLoginModule="WLDeviceAutoProvisioningLoginI
                <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
            </loginModule>
        </loginModules>
```

Single sign-on inconsistent state

Avoid conflicts in the single sign-on configuration of a login module. Such
conflicts cause inconsistency in the single sign-on state of the login
module, and can lead to unexpected results.

A conflict can exist between the configuration of a loginModule element
and the configuration of a mobileSecurityTest element. This can happen
when you enable single sign-on of a login module in the loginModule
element and then disable single sign-on for the same login module (by
using it in a mobileSecurityTest without specifying sso="true" for the
realm of this loginModule). For example:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm"/>
    </mobileSecurityTest>
</securityTests>
<realms>
    <realm name="SampleAppRealm" loginModule="StrongDummy">
        <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
    </realm>
</realms>
<loginModules>
    <loginModule name="StrongDummy" ssoDeviceLoginModule="WLDeviceAutoProvisioningLoginMod
        <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
    </loginModule>
</loginModules>
```

Another case of conflict can happen between different mobileSecurityTest
elements, when two mobileSecurityTest elements use the same login
module, with conflicting values for the **sso** attribute. For example, when
using the same realm with conflicting **sso** enablement state in two
mobileSecurityTest elements:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm"/>
    </mobileSecurityTest>
    <mobileSecurityTest name="FormTestWithSso">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
</securityTests>
```

Another example, in which the same login module is used for different
realms with conflicting SSO enablement states in two mobileSecurityTest
elements:

```
<securityTests>
    <mobileSecurityTest name="FormTest">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealm"/>
    </mobileSecurityTest>
    <mobileSecurityTest name="FormTestWithSso">
        <testDeviceId provisioningType="none"/>
        <testUser realm="SampleAppRealmWithSso" sso="true"/>
    </mobileSecurityTest>
</securityTests>
```

```
<realms>
    <realm name="SampleAppRealm" loginModule="StrongDummy">
        <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
    </realm>
    <realm name="SampleAppRealmWithSso" loginModule="StrongDummy">
        <className>com.worklight.core.auth.ext.FormBasedAuthenticator</className>
    </realm>
</realms>
```

What to do next

Additional configuration and settings are required when you use a reverse proxy. Either of the following options allow device SSO to work with a reverse proxy.

Device single sign-on with the IBM Security Access Manager Web reverse proxy:

Additional configuration and settings are required when you use the IBM Security Access Manager Web reverse proxy.

You can configure the IBM Security Access Manager Web reverse proxy or IBM Security Access Manager WebSEAL when you enable Worklight device SSO to delegate user authentication to the Worklight Device SSO realm. For more information about the required configurations and samples, see IBM Security Access Manager for IBM Worklight.

This option is supported on all mobile platforms and you can use the IBM Security Access Manager features such as Risk-Based Access (RBA), Context-Based Access (CBA), Strong Authentication (One-time password), and Identity aware applications (OAuth) to further enhance security.

Configuring device single sign-on with a reverse proxy:

Additional configuration and settings are required when you use a reverse proxy.

Before you begin

Ensure that you configured device single sign-on as explained in "Configuring device single sign-on" on page 912.

About this task

Device SSO and reverse proxies

> Device single sign-on with a reverse proxy can also be achieved with the "Simple data sharing" on page 925 feature. The Simple Data Sharing feature allows a set of applications to share authentication cookies that allow access through the reverse proxy and delegate authentication to the Worklight Server Device SSO realm.

> The Simple Data Sharing feature is supported only on iOS and Android devices.

> With the Simple Data Sharing feature, you can tell the Worklight client runtime environment to share credentials among applications in the same Worklight application family. Because you are working with security tokens, you must ensure that access to the applications is protected by other mechanisms. For example, ensure that the device is not jailbroken,

and that the device is password-protected. For more information, see "Simple data sharing limitations and special considerations" on page 931.

The following steps show how to configure device single sign-on with a reverse proxy.

Procedure

1. Enable the Simple Data Sharing feature as explained in "Enabling the Simple Data Sharing feature" on page 927.
2. For hybrid applications, follow these steps.
   a. Ensure that you select the Worklight device SSO option.
   b. Specify a comma-separated list of cookie names that you want IBM Worklight Foundation to remember and share among the applications in your specified family.

3. For native applications, follow these steps.
   a. Add the `wlShareCookies` property in the Worklight properties file.
   b. Specify a comma-separated list of cookie names that you want IBM Worklight Foundation to remember and share among the applications in your specified family.

   `wlShareCookies = PD-S-SESSION-ID`

   Each application in the Worklight family must be enabled for simple data sharing, and must also specify the cookie, which it agrees to share and reuse. For example, you can specify any one of the `PD-*SESSION-ID` cookies for IBM Security Access Manager or the `Ltpatoken` or `Ltpatoken2` cookies for IBM WebSphere DataPower.

Results

You have configured device single sign-on with a reverse proxy.

Using SSO between IBM Worklight Foundation and external services

You can use single sign-on (SSO) between IBM Worklight Foundation and external services by making use of the Worklight security framework to protect the external services.

About this task

Worklight Server acts as an authorization server and issues an access token that can be validated by the external service. The client application requests the access token from IBM Worklight Foundation via the token endpoint and sends it to the external services.

The scope of the access token is a security test defined inside a Worklight project. Each scope has a timeout property that determines the lifetime of the token. This defines the time for which an issued token remains valid. Once the timeout expires, the token is rejected and a new one needs to be requested from the server.

Note: If Worklight Server and the external service are visible to the client through different domains, the following restrictions apply:
- The solution is inappropriate for web environment.
- Web preview for mobile environment does not work.

Procedure

1. Configure the Worklight project.

    a. Configure the scope for the access token.

    The scope of an access token must be a predefined security test in a Worklight project. The security test is configured in *WL_Project*/server/conf/authenticationConfig.xml. The default lifetime for each token is 60 seconds, which you can override by adding to the security test the AccessTokenExpirationSec attribute. For example, if you want to configure a security test called SampleSecurityTest with a lifetime of 15 seconds, you edit the authenticationConfig.xml file in either of the following ways:

    From Source view:
    ```
    <securityTests>
        <customSecurityTest name="SampleSecurityTest" AccessTokenExpirationSec="15">
            <test realm="SampleRealm" isInternalUserID="true"/>
        </customSecurityTest>
    </securityTests>
    ```

    From Design view:
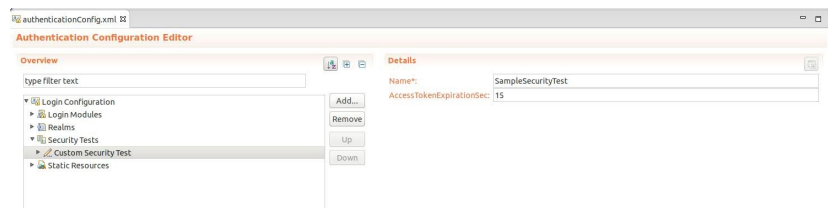
    b. Use a keystore.

    Create and use your own keystore, and configure the Worklight Server to use it. For information about how to create a keystore in an unrelated context, see "Configuring device auto provisioning" on page 905.

    Attention: Using the default keystore is not secure.

2. Configure the external service.

    To ensure that your external service accepts the access token, you must add a validation library to your service that can validate the token with either online or offline validation. Two libraries are provided for this purpose:
    - worklight-access-token-validator.jar (Java lib)
    - worklight-access-token-validator.tgz (Node.js module)

For Worklight Server installation

The libraries can be found under: *install_dir*/WorklightServer/
external-server-libraries.

For Worklight Studio

Upon creating a new project, the libraries can be found under:
project_dir/externalServerLibraries .

You must use one of the following options:

- **Option i**: Configure the external service by using Java.

  The purpose of this module is to allow offline validation of access tokens
  generated by Worklight Server for Java web applications.

  To use the Java library, two files are needed:

  – Certificate of Worklight Server.

    Export the certificate from the keystore of the Worklight Server. You can
    do this with the Java keytool.

  – worklight-access-token-validator.jar.

  You can use either a servlet filter, or use the Java-supplied API:

  Using a servlet filter

  Add this JAR to the class path of your web application, and use the
  filter class com.worklight.security.WLAccessTokenValidationFilter
  as shown in the following example. Assume the values in the
  following table:

Table 146. Example servlet filter parameter values

Parameter	Value	Explanation
Filter name	FilterName	Choose an arbitrary name for the filter.
URL	/some/protected/url	Prefix for all the resources you want to protect.
Scope [Optional]	securityTestName	Name of the security test, as defined in the authenticationConfig.xml file, which is needed to authenticate against in order to gain access to the protected resources.
CertificatePath	certificateLib/ WorklightServerCeriificate.cert	Path to the certificate of the Worklight Server relative to the WEB-INF folder.

Assuming the parameter values in the previous table, this is the
addition needed for the web.xml of your external server:

```
<web-app ...>
...
<filter>
    <filter-name>FilterName</filter-name>
    <filter-class>com.worklight.security.WLAccessTokenValidationFilter</filter-class>
    <init-param>
        <param-name>worklightCertificateFile</param-name>
        <param-value>certificateLib/WorklightServerCeriificate.cert</param-value>
    </init-param>
    <init-param>
        <param-name>scope</param-name>
        <param-value>securityTestName</param-value>
```

```
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>FilterName</filter-name>
        <url-pattern>/some/protected/url</url-pattern>
    </filter-mapping>
    ...
</web-app>
```

After successful validation, the filter updates the `ClientContext` object that can be used by the service to access user, application, or device identities contained in the access token. This is an example of `ClientContext` usage:

```
ClientContext context = ClientContext.getInstance();
String appId = context.getApplcation();
String userId = context.getUser();
String deviceId = context.getDevice();
```

Using the Java-supplied API

The following interface is exposed:

```
package com.worklight.common.security.oauth;
public interface IAccessToken {
  public String getUserIdentity();
  public String getDeviceIdentity();
  public String getApplicationIdentity();
  public String getVersion();
  public String getScope();
}
```

The class `AccessTokenParse` provides the following API in order to get an instance that implements this interface:

```
public static IAccessToken parseToken(final String tokenStr, final PublicKey serverPub
public static IAccessToken parseToken(final String tokenStr, final PublicKey serverPub
```

Both methods check the validity of the token (correctly formatted token and issued by Worklight Server for the given public key), and that the token has not expired.

The only difference between the two methods is that the first method also validates that the token was issued for the given scope. The public key needs to be taken from the certificate: For example:

```
CertificateFactory cf = CertificateFactory.getInstance("X.509");
X509Certificate cert = (X509Certificate) cf.generateCertificate(certInputStream);
wlPublicKey = cert.getPublicKey();
```

The interface can be used in the following way:

```
try {
  IAccessToken iAccessToken = null;
  if (scope != null) {
    iAccessToken = AccessTokenParser.parseToken(wlToken, wlPublicKey, scope);
  } else {
    iAccessToken = AccessTokenParser.parseToken(wlToken, wlPublicKey);
  }

  String userId = iAccessToken.getUserIdentity();
  String deviceId = iAccessToken.getDeviceIdentity();
  String appId = iAccessToken.getApplicationIdentity();

  ...
  ...
```

- **Option ii**: Configure the external service by using Node.js.

The purpose of this node module is to allow offline validation of access tokens generated by Worklight Server for Node.js server.

a. Certificate of Worklight Server.

   In order to get the certificate, you need to generate a `.pem` certificate from the keystore. One possible way to do so is with Java's keytool. For example, from bash, creating the `.pem` certificate from jks keystore:

   ```
   keytool -exportcert -keystore $KEYSTORE_FILE -alias $CERTIFICATE_ALIAS -rfc > $OUTPUT_FI
   ```

   You then have to pass the content of the generated (PEM-formatted) certificate file as input to the node module, which allows you to validate tokens created with the same certificate.

   Note: The expiration of the token is checked against the local computer time, so ensure your clock is synchronized (preferably using an NTP server).

b. `worklight-access-token-validator.tgz`

   You will need to install the module with:

   ```
   npm install tgz file
   ```

   Using this module gives you a function that requires certificate (mandatory) and a scope (optional). Once called with these parameters, you have an object with the following functions:

Table 147.

Function	Description
`validate(token, callback)`	A function that validates if the token provided is a valid Worklight access token. If the `scope` parameter is given upon initialization, the function also validates that the token is for the required scope. • token • callback - `function(errorObject, authenticationData)` ** both objects described below
`validateAuthorizationHeader(authHeader, callback)`	A helper function. Allows developer to use it without having to parse the 'Authorization' header to retrieve the token • authHeader • callback - as described above

For example:

```
// Load the certificate from a PEM encoded file
var cert = require("fs").readFileSync('cert.pem');
// The scope to mandate (can be null, in which case the token is only checked for a suit
var scope = "WorlightSecurityTest";
// Create a reusable validator
var worklightValidator = require("worklight-access-token-validator")(cert, scope);
worklightValidator.validate(token, function(error, payload) {
  if (error) {
    // Token is invalid, send appropriate response to user
    response.writeHead(error.httpStatus,{"WWW-Authenticate":error.wwwAuthenticateHeader}
  } else {
    // Token is valid, proceed with request
  }
});
```

The following table lists the fields that the error object in the callback method contains:

Table 148. Error object fields

Field	Explanation
err	Contains the error code, which can be one of the following: • 'invalid_token' • 'missing_token' • 'insufficient_scope'
errMsg	Contains the human-readable description of the error reason.
httpStatus	HTTP status to use when responding to the access token sender.
wwwAuthenticateHeader	Content of the 'WWW-Authenticate' header that should be responded to the access token sender.

The authentication data in the callback method contains the following fields:

Table 149. Fields in the authentication data in the callback method

Field	Explanation
version	Version of the token.
scope	Security test that the token authenticated.
expiration	Time (in milliseconds) since epoch when the token expires.
data	Object with the following fields. **user_id [optional]** Authenticated user **device_id [optional]** device id as known by worklight server **application_id** identity of the app

- **Option iii**: Configure the external service by using a validation endpoint.

  When offline validation is inappropriate, you can use online validation of the access tokens with the /oauth/validation endpoint. This endpoint provides the signature validation, expiration check, and optional scope validation.

  For this example (in pseudo-code), the optional scope parameter will not be passed to the endpoint.

```
filter(request):
  header = getHeader(request, 'Authorization')
  // header should have the format: "Bearer <token>"
  token = parseHeader(header)

  // Call validation endpoint as described
  res = callValidationEndpoint(token)
  if (res.code != SUCCESS) {
    // Token not validated, reponse can be sent as is to caller
    sendResponse(res)
    exit
  }

  // If successful
```

```
payload = res.body
scope = payload.scope
userId = payload.data.user_id

// Continue with scope checking | passed filter, return payload/data
```

What to do next

Consider using the client-side API features that support the use of Worklight access tokens.

Client-side API

The WL.Client API offers built-in support for using Worklight access tokens for the following platforms:

- Hybrid: JavaScript
- Android
- iOS

You can use the methods included in this API in the following ways:

Obtaining and caching a token for a specified scope

WL.Client requests a new token from the Worklight Server. To obtain the token, the client must be authenticated in all realms of the requested scope (which is represented by a security test in Worklight server's AuthenticationConfig.xml file). Thus, calling this method might trigger an authentication sequence for all realms for which authentication is still required.

This method is asynchronous in all platforms. It does not return a value, but instead triggers a response handler. Note that there is no need to parse the response from the server in the response handler. The token is automatically parsed and cached inside WL.Client and can be retrieved by using the following method:

Getting the last obtained access token

WL.Client returns the last access token for a certain scope. Alternatively, if no scope is provided, the last obtained token is returned. This is useful when an application is only using one scope.

The scope is represented as a string and should be added to the "Authorization" header of the request to the protected external server, preceded by "Bearer". The following example demonstrates how you might do this when issuing an ajax request:

```
var token = WL.Client.getLastAccessToken();
  $.ajax({
    type : "GET",
    url : MY_URL,
    headers : {"Authorization" : "Bearer " + token}
    })
```

Getting the required scope from the external service response

When a request to the external service fails, WL.Client is able to identify whether the failure is related to access token issues (for example, the token does not match the required scope, the token has expired), and will return the name of the scope which is required in order to access the service. In this case, obtaining a

new token for the returned scope is required. If the error is not related to access token issues, this method returns `null`.

Suggested use of client-side API

This JavaScript example shows how to use the client-side API when accessing an external service:

```
function callProtectedRestAPI(retries) {

  // We want to be able to call this method recursively, since in some cases
  // we need to obtain a new token, and try a second time.
  if (retries == 0) {
  return;
  }

  // Get the last obtained access token.
  // On the first call, the token may be null.
  var token = WL.Client.getLastAccessToken();
  var headersObject = (token != null) ? {"Authorization" : "Bearer " + token} : {};

  $.ajax({
    type : "GET",
    url : MY_EXTERNAL_SERVER_URL,
    headers : headersObject

  }).done(function(response) {
    showResult(response);
  }).fail(
    function(response) {
      // We need to extract this header from
      // the response in order to know the scope.
      var header = response.getResponseHeader("WWW-Authenticate");
      var scope =
        WL.Client.getRequiredAccessTokenScope(response.status,header);
      if (scope != null) {

        // The failure is related to the access token. Get a new one.
        WL.Client.obtainAccessToken(
          scope, getTokenSuccess,getTokenFailure);
      } else {
        showErrorResult("request failed");
      }
  });

  function getTokenSuccess(response) {

    // We obtained a token. try to access the external server one more time.
    callProtectedRestAPI(retries - 1);
  };

  function getTokenFailure(response) {
  showErrorResult(response);
  };
}
```

WL.Client API reference information

For more information about the WL.Client API, see the following sections:

- WL.Client JavaScript API
- WL.Client iOS API
- WL.Client Android API

Exposed endpoints

This feature exposes two new endpoints of the Worklight Server:

/oauth/token

Used from the Worklight app in order to authenticate for a desired scope. This endpoint should be used with one of the client-side APIs provided. (See section WL.Client API reference information earlier in this topic for a list of links to WL.Client API reference information.)

Return:

- If authentication fails, return code and appropriate error message is returned as defined by Oauth 2.0 RFC.
- If authentication flow is completed successfully, returns a valid access token.

/oauth/validation

This endpoint can be used to validate an access token which was created by the same Worklight Server. A valid request will have the following properties:

Table 150. Validation request properties

Property	Description
URL	*<context-root>*/oauth/validation
Method	POST
Parameter: token (mandatory)	The token in question.
Parameter: scope (optional)	The scope that protects the resource.

Regardless of whether scope is supplied or not, the endpoint makes sure that the token is valid. If the optional parameter scope is provided, the endpoint verifies that the token is provided for the required scope.

Return:

- If validation fails, return code & appropriate header will be returned as defined by Oauth 2.0 RFC for the Bearer token.
- If successful, the payload of the token is returned to user. The payload is a JSON object, and as of token version WL1.0, its format is:

```
{
  version: 1.0 (version of token)
  scope: <The security test that the token authenticated against>
  expiration: <Time in msec since epoch when token will be expired>
  data: {
    user_id: <authenticated user>
    device_id: <device id as known by Worklight server>
    application_id: <identity of the app>
  }
}
```

Simple data sharing

Learn about the Simple Data Sharing feature.

Simple data sharing overview

Learn about the Simple Data Sharing feature.

The Simple Data Sharing feature makes it possible to securely share lightweight information among a family of applications on a single device. This feature uses native APIs that are already present in the different mobile SDKs to provide one

unified developer API. This Worklight API abstracts the different platform complexities, making it easier for developers to quickly implement code that allows for inter-application communication.

This feature is supported on iOS and Android for both hybrid and native applications.

After you enable the Simple Data Sharing feature, you can use the provided hybrid and native APIs to exchange simple string tokens among a family of applications on a device.

When used with other features like the Worklight device single sign-on (SSO) or User Certificate Authentication features, the Simple Data Sharing feature enhances the ability of these features to share security credentials among applications in the same family. For example, you can share user authentication cookies among a family of applications to allow device SSO to work when a reverse proxy is used. It also enables the User Certificate Authentication feature to provision an X.509 user certificate to a family of applications on a device.

For more information about device SSO with a reverse proxy, see "Configuring device single sign-on with a reverse proxy" on page 916.

For more information about user certificate group support, see "Configuring user certificate authentication for a group of applications" on page 1462.

Simple data sharing general terminology

Learn about simple data sharing general terminology.

Worklight application family

An application family is a way to associate a group of applications which share the same level of trust. Applications in the same family can securely and safely share information with each other.

To be considered part of the same Worklight application family, all applications in the same family must comply with the following requirements:
- Specify the same value for the application family in the application descriptor.
  - For iOS applications, this requirement is synonymous to the access group entitlements value and the `wlAppFamily` value in the `worklight.plist` file.
  - For Android applications, this requirement is synonymous to the `sharedUserId` value in the Android manifest file.
    - For Android, the name must be in the *x.y* format.
    -

      Note: Enabling or changing the Worklight application family settings require prior Android applications to be uninstalled. Upgrading an application that modified its `sharedUserId` is not allowed by the Android operating system for security reasons.
- Applications must be signed by the same signing identity. This requirement means that only applications from the same organization can use this feature.
  - For iOS applications, this requirement means the same Application ID prefix, provisioning profile, and signing identity is used to sign the application.

- For Android applications, this requirement means the same signing certificate and key.

Aside from the IBM Worklight Foundation provided APIs, applications in the same Worklight application family can also use the data sharing APIs that are available through their respective native mobile SDK APIs.

String tokens

Sharing string tokens across applications of the same Worklight application family can now be accomplished through hybrid or native iOS and Android applications through the Simple Data Sharing feature.

String tokens are considered simple strings, such as passwords or cookies. Using large strings results in considerable performance degradation.

Consider encrypting tokens when you use the APIs for added security. For more information, see "JSONStore security utilities" on page 837.

Enabling the Simple Data Sharing feature

Learn how to enable the Simple Data Sharing feature.

Enabling the Simple Data Sharing feature for hybrid applications

Update the application descriptor to enable the Simple Data Sharing feature on hybrid applications.

About this task

To enable simple data sharing, you must modify the application descriptor.

Procedure

1. Add an Android or iOS environment.
2. Add the **Simple Data Sharing** option.
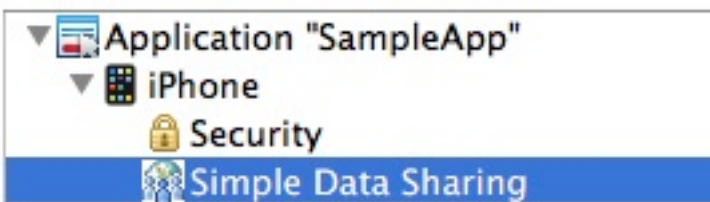
3. Enable the Simple Data Sharing option and specify an application family name.

> **Note:** For Android, the application family name must be in the form of `com.xx.yy`.

Note: Enabling or changing the Worklight application family settings require prior Android applications to be uninstalled. Upgrading an application that modified its `sharedUserId` is not allowed by the Android operating system for security reasons.

4. Save.

5. Build all environments.

6. Ensure that applications that are part of the same family are signed by the same signing credentials.

7. For Android environments, follow these steps.

   a. Before you install the newly built application on the device, uninstall any prior applications from the device that were using a different family name value.

   b. Install the newly built application on the device.

8. For iOS environments, follow these steps.

   a. Ensure that applications that are part of the same family share the same Application ID prefix. For more information, see Managing Multiple App ID Prefixes in the iOS Developer Library.

9. Repeat the steps for all applications that you want to make part of the same application family.

Results

You can now use the Simple Data Sharing JavaScript APIs to share simple strings among the group of applications in the same family. For more information, see the Simple Data Sharing JavaScript APIs in the `WL.Client` class.

Enabling the Simple Data Sharing feature for iOS native applications
Update iOS native applications to enable the Simple Data Sharing feature.

Before you begin

For more information about how to develop iOS native applications, see "Developing native applications for iOS" on page 616.

Note: Only applications from the same organization can use this feature.

About this task

To enable simple data sharing, you must modify your iOS native application.

Procedure

1. Enable the Simple Data Sharing option by specifying the application family name in the `worklight.plist` file with the `wlAppFamily` property.

wlAppFamily	String	myGroup1

2. In Xcode, add a Keychain Access Group with the same name as your `wlAppFamily`.

Keychain Groups: worklight.group

                  myGroup1

                  + −
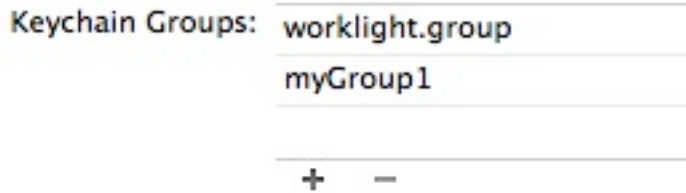
The application-identifier entitlement must be the same for all applications in your family.

Note: By default, Worklight 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. Ensure that applications that are part of the same family share the same Application ID prefix. For more information, see Managing Multiple App ID Prefixes in the iOS Developer Library.
4. Save and sign applications. Ensure that all applications in this group are signed by the same iOS certificate and provisioning profiles.
5. Repeat the steps for all applications that you want to make part of the same application family.

Results

You can now use the native Simple Data Sharing APIs to share simple strings among the group of applications in the same family. For more information, see the Simple Data Sharing Objective-C APIs in the WLSimpleDataSharing class.

Enabling the Simple Data Sharing feature for Android native applications
Update Android native applications to enable the Simple Data Sharing feature.

Before you begin

For more information about how to develop Android native applications, see "Developing native applications for Android" on page 621.

About this task

To enable simple data sharing, you must modify your Android native application.

Procedure
1. Enable the Simple Data Sharing option by specifying the application family name as the android:sharedUserId element in the manifest tag of your Android manifest file.
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
          package="com.myApp1"
          android:versionCode="1"
          android:versionName="1.0"
          android:sharedUserId="com.myGroup1">
```
The sharedUserId is equivalent to your wlAppFamily name.

> **Note:** Enabling or changing the Worklight application family settings require prior Android applications to be uninstalled. Upgrading an application that modified its `sharedUserId` is not allowed by the Android operating system for security reasons.

2. Ensure that applications that are part of the same family are signed by the same signing credentials.
3. Uninstall any prior versions of the applications that did not specify a `sharedUserId` or that used a different `sharedUserId`.
4. Install the application on the device.
5. Repeat the steps for all applications that you want to make part of the same application family.

Results

You can now use the native Simple Data Sharing APIs that are provided to share simple strings among the group of applications in the same family. For more information, see the Simple Data Sharing Java APIs in the `WLSimpleDataSharing` class.

Simple data sharing API concepts

Learn about simple data sharing API concepts.

Sharing string tokens across applications of the same Worklight application family can be accomplished through iOS and Android for both hybrid and native applications. This API is meant for sharing simple strings securely.

The Simple Data Sharing APIs allow any application in the same family to set, get, and clear key-value pairs from a common place. The Simple Data Sharing APIs are similar for every platform, and provide an abstraction layer, hiding the complexities that exist with each native SDK's APIs, making it easy to use.

The following examples show how you can set, get, and delete tokens from the shared credential storage for the different environments.

Hybrid applications

```
WL.Client.setSharedToken({key: myName, value: myValue})
WL.Client.getSharedToken({key: myName})
WL.Client.clearSharedToken({key: myName})
```

For more information about the hybrid APIs, see the `getSharedToken`, `setSharedToken`, and `clearSharedToken` functions in the `WL.Client` class.

iOS native applications

```
[WLSimpleDataSharing setSharedToken: myName value: myValue];
NSString* token = [WLSimpleDataSharing getSharedToken: myName]];
[WLSimpleDataSharing clearSharedToken: myName];
```

For more information about the native iOS APIs, see `WLSimpleDataSharing` Class Reference.

Android native applications

```
WLSimpleSharedData.setSharedToken(myName, myValue);
String token = WLSimpleSharedData.getSharedToken(myName);
WLSimpleSharedData.clearSharedToken(myName);
```

For more information about the native Android APIs, see Class
`WLSimpleDataSharing`.

Simple data sharing troubleshooting

Find information to help resolve issues that you might encounter when you use
the Simple Data Sharing feature.

Table 151. Troubleshooting the Simple Data Sharing feature. This table lists possible
problems and actions to take to troubleshoot the Simple Data Sharing feature.

Problem	Actions to take
Unable to access shared data when you use the Simple Data Sharing APIs.	Ensure that all applications in the same family are all redeployed under the same Worklight application family name. For more information, see "Enabling the Simple Data Sharing feature" on page 927.
Android: Application fails to install.	1. Ensure that all applications that are part of the same Worklight application family name, are also signed by the same signing identity. 2. Uninstall previous versions of applications that have a different Worklight application family or `wlSharedUserID`, or are signed by a different signing identity.
Unable to get Worklight device SSO to work with a reverse proxy.	1. Ensure that you enabled the Simple Data Sharing feature. For more information, see "Enabling the Simple Data Sharing feature" on page 927. 2. Ensure that all applications in the same family specified the necessary reverse proxy authentication cookie. For more information, see "Configuring device single sign-on with a reverse proxy" on page 916.
Unable to specify cookie or user certificate sharing.	You must first enable the Worklight Simple Data Sharing feature and specify a Worklight application family before you can enable device SSO or user certificate authentication sharing options. For more information, see "Enabling the Simple Data Sharing feature" on page 927.

Simple data sharing limitations and special considerations

Learn about the limitations and special considerations of the Simple Data Sharing
feature.

Security considerations

Because this feature allows for data access among a group of applications, special
care must be taken to protect access to the device from unauthorized users.
Consider the following security aspects:

Device Lock

For added security, ensure that devices are secured by a device password, passcode, or pin, so that access to the device is secured if the device is lost or stolen.

Jailbreak Detection

Consider using a mobile device management solution to ensure that devices in your enterprise are not jailbroken or rooted.

Encryption

Consider encrypting any tokens before you share them for added security. For more information, see "JSONStore security utilities" on page 837.

Size limit

This feature is meant for sharing of small strings, such as passwords or cookies. Be cognizant not to abuse this feature, as there are performance implications with such attempts to encrypt and decrypt or read and write any large values of data.

Maintenance challenges

Android developers must be aware that enabling this feature, or changing the application family value, results in their inability to upgrade existing applications that were installed under a different family name. For security reasons, Android requires prior applications to be uninstalled before applications under a new family name can be installed.

Developing globalized hybrid applications

To develop globalized hybrid applications, learn about globalization in JavaScript frameworks and IBM Worklight Foundation, 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 Foundation 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 Foundation and how to globalize web services and push notifications.

Globalization in JavaScript frameworks

You can use several JavaScript frameworks to globalize your applications: Dojo, jQuery, and Sencha Touch.

You can use the capabilities of different JavaScript frameworks to globalize your applications. Dojo, jQuery, and Sencha Touch each provide globalization functions that are based on resource bundles and resource files, and they can switch to different resource bundles based on current locale information. In addition, Dojo also provides string and date format utilities that are based on user locale information.

Dojo globalization framework

You can use the Dojo globalization framework to globalize your application.

The following example application demonstrates how to use the Dojo Mobile JavaScript API to construct a globalized application with a native look and feel.

Dojo Mobile provides globalization functions for detecting locale, loading and accessing resource bundles, and simple formatting utilities for culture-sensitive display. Figure 1 and Figure 2 show pages of the application that display the resource bundles loaded and the string format that is determined by the user preferences on the device.
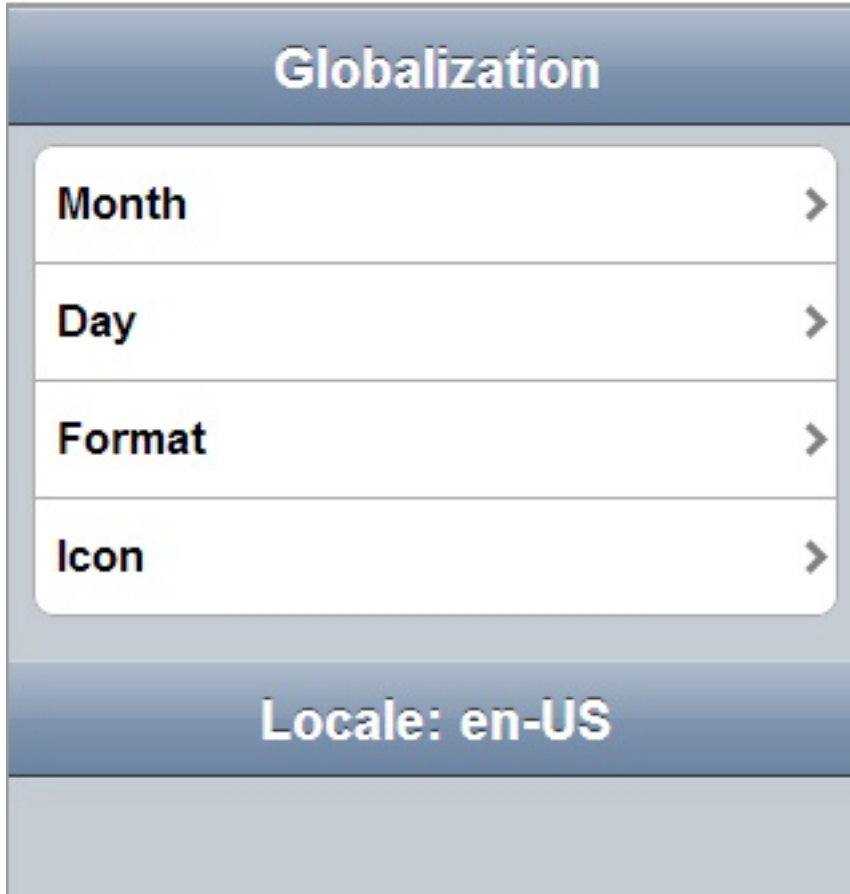
Figure 113. Dojo globalization application

Figure 114. Dojo cultural formatting

In the example, the Dojo library is loaded as shown in Listing 1: Including Dojo Mobile. The required modules must be loaded before you can use the Dojo globalization API.

Listing 1: Including Dojo Mobile

```
<script type="text/javascript">
  var dojoConfig = {
    parseOnLoad: false,
    packages: [{
      name: "resource",
      location: "../../bundles"
    }]
  };
</script>
<script type="text/javascript" src="libs/dojox/mobile/deviceTheme.js"></script>
<script type="text/javascript" src="libs/dojo/dojo.js"></script>
```

Figure 3 shows how to load Dojo resource bundles by defining a package that maps the location of resource files within your hybrid application to a package name.
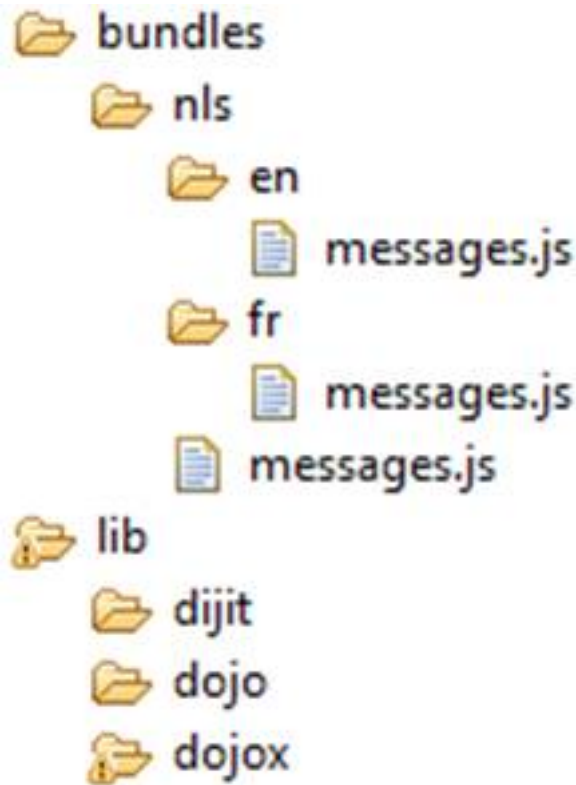
Figure 115. Dojo resource bundle structure

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"><
  <ul data-dojo-type="dojox.mobile.RoundRectList">
    <li id="months_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="months" callback="getMon
    <li id="days_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="days" callback="getDays" l
    <li id="formats_choice" data-dojo-type="dojox.mobile.ListItem" moveTo="formats" callback="getF
    <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_fo
</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_ic
</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_
</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_days"
</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_fo
</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.
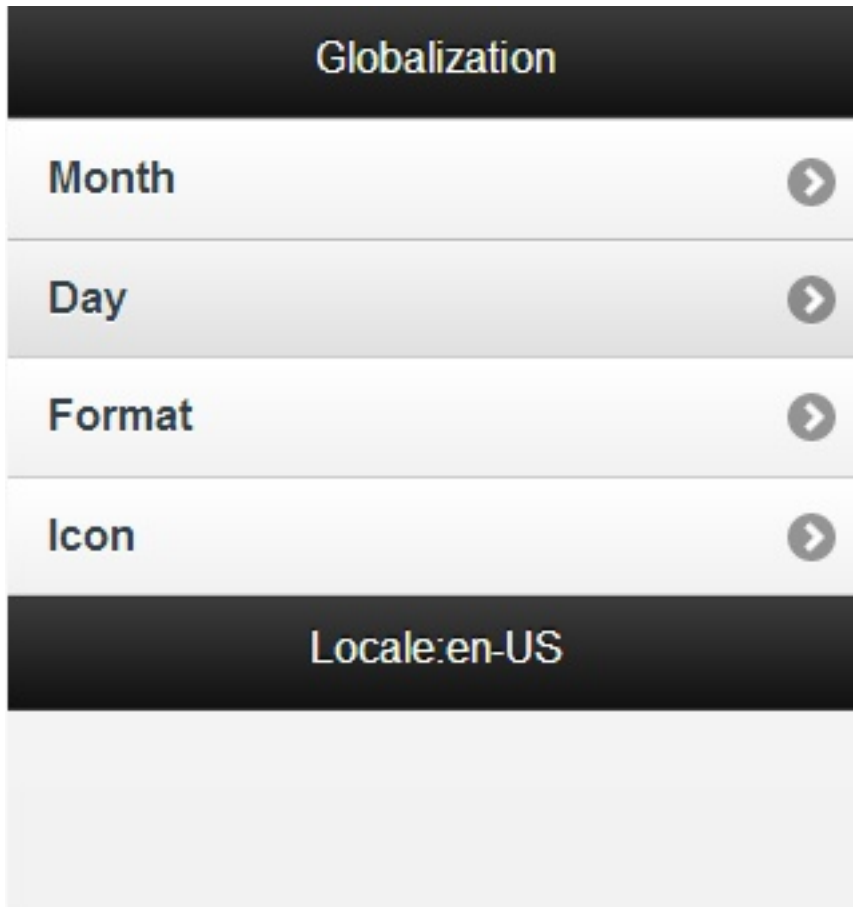
Figure 116. jQuery globalization application

Listing 1: Load Cordova, jQuery mobile, and jQuery globalization plug-in

This listing shows the scripts for loading Cordova, jQuery mobile, and the
`jquery.i18n.properties-1.0.9.js` jQuery globalization plug-in.

```
<script
  type="text/javascript"
  src="js/CordovaGlobalization.js">
</script>
<script
  type="text/javascript"
  src="js/messages.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.mobile-1.1.1.min.js">
</script>
<script
  type="text/javascript"
  src="js/jquery.i18n.properties-min-1.0.9.js">
</script>
```

The resource bundle structures in jQuery and Dojo are different. Dojo resource files
have the same file name but are in separate folders corresponding to the locale
name. jQuery resource files are in one folder but the file names include the locale
information. Figure 2 shows the structure of the jQuery resource files.
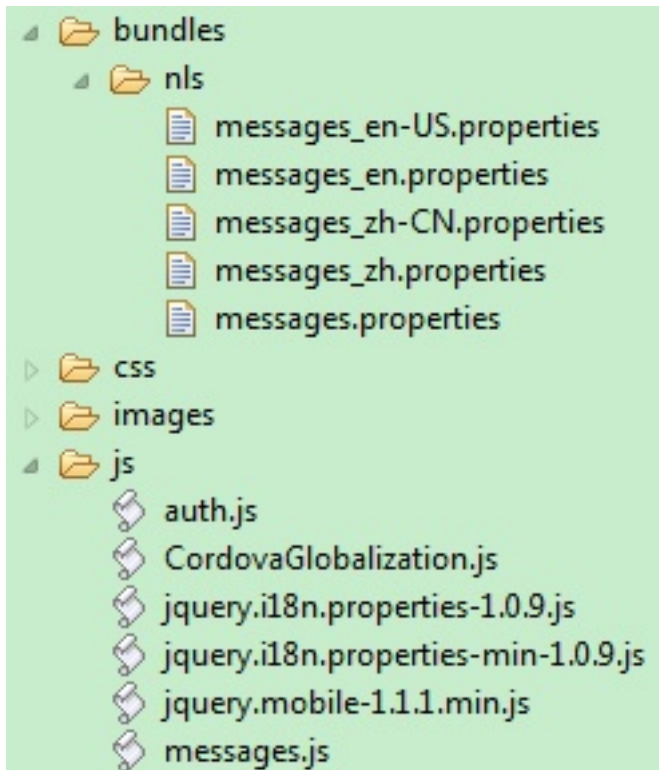
Figure 117. jQuery resource bundle structure

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 GitHub.

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.
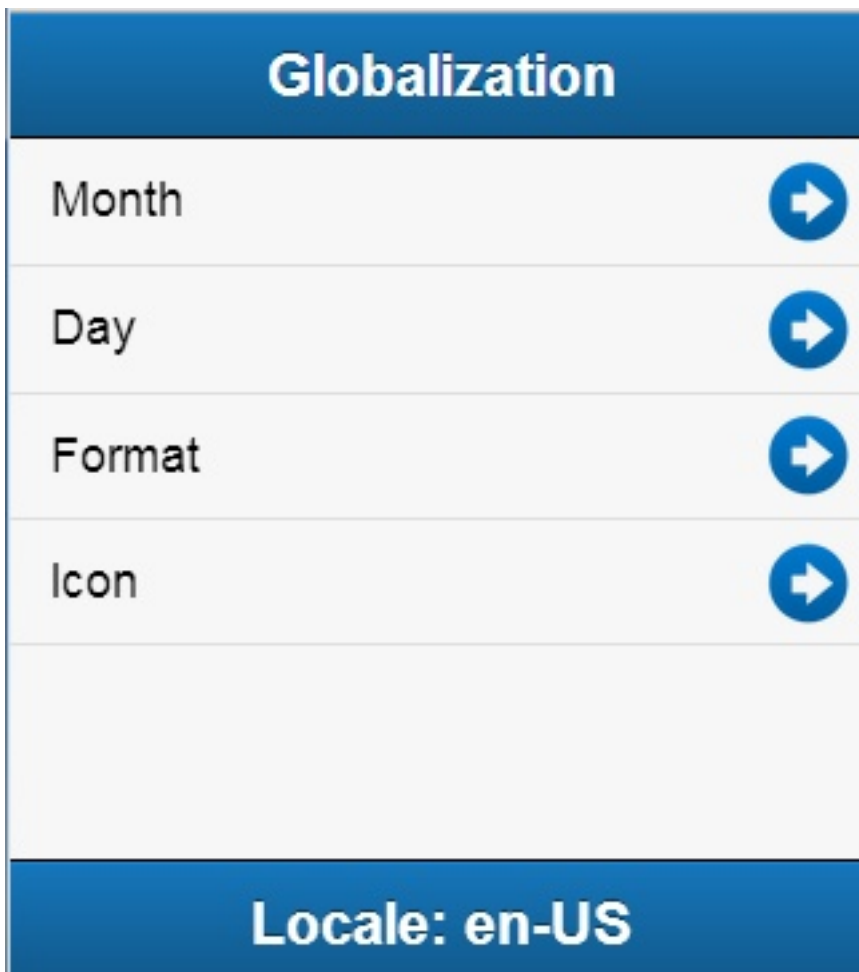
Figure 118. Sencha Touch globalization application

Listing 1: Load Sencha Touch and globalization plug-in

This listing shows the scripts for loading Sencha Touch and the `Ext.i18n.bundle-touch` globalization plug-in.

```
<script src="js/sencha-touch-all.js"></script>
<script>
  Ext.Loader.setConfig({
    enabled: true,
    paths: {
      'Ext.i18n': 'js/i18n',
```

```
                'patch': 'js/patch'
            }
        });
    </script>
    <script src="js/SenchaGlobalization.js"></script>
    <script src="js/messages.js"></script>
    <script src="js/auth.js"></script>
```

Figure 2 shows the structure of the Sencha Touch resource files.
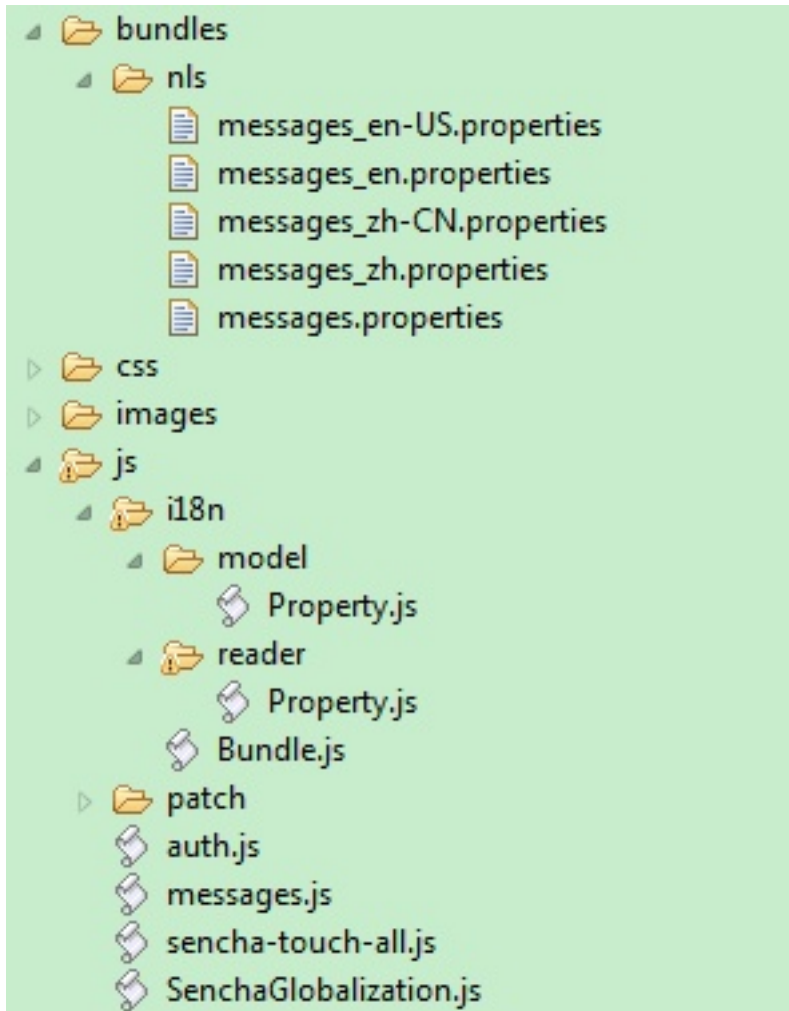
Figure 119. Sencha Touch resource bundle structure

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 Foundation

IBM Worklight Foundation 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 152. 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 153. 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 154. Listing 2: Month names

```
function getMonths(){
  g11n.getDateNames(function(data){
    var items = data.value;
    var monthsView = document.getElementById('monthsView');
    for (var i = 0; i < items.length; i++) {
      monthsView.append('<li>' + items[i] + '</li>');
    }
  },
  function(code){
    alert("Error: " + code);
  });
};
```

Figure 120. Using Cordova to show locale-based months

Table 155. 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"}
  );
}
```
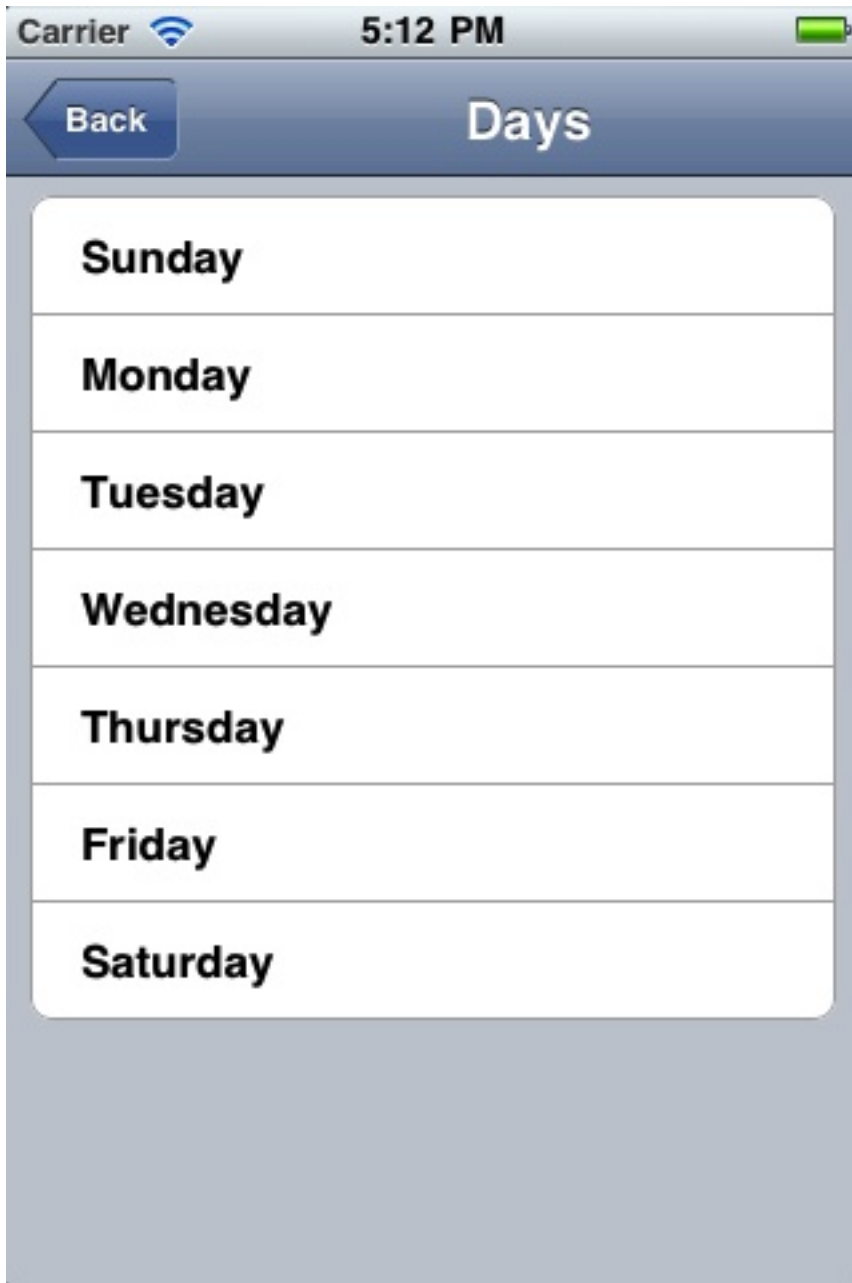
Figure 121. Using Cordova to show the days of week

Table 156. Listing 4: Formatting current date

```
function getFormats(){
  var formatsView = document.getElementById('formatsView');
  g11n.dateToString(
    new Date(),
    function(date){
      formatsView.append('<li>' + date.value + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {selector: "date", formatLength: "full"}
  );
  g11n.getDatePattern(
    function(date){
      formatsView.append('<li>' + date.pattern + '</li>');
      var timeZone = date.timezone;
      formatsView.append('<li>' + timeZone + '</li>');
      var offset = date.utc_offset;
      formatsView.append('<li>' + offset + '</li>');
      var dstoffset = date.dst_offset;
      formatsView.append('<li>' + dstoffset + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {selector: "date", formatLength: "full"}
  );
  g11n.isDayLightSavingsTime(
    new Date(),
    function(date){
      var dst = date.dst;
      formatsView.append('<li>' + dst + '</li>');
    },
    function(code){
      alert("Error: " + code);
    }
  );
  g11n.numberToString(
    1234.56,
    function(number){
      formatsView.append('<li>' + number.value + '</li>');
    },
    function(code){
      alert("Error: " + code);
    },
    {type: "decimal"}
  );
}
```
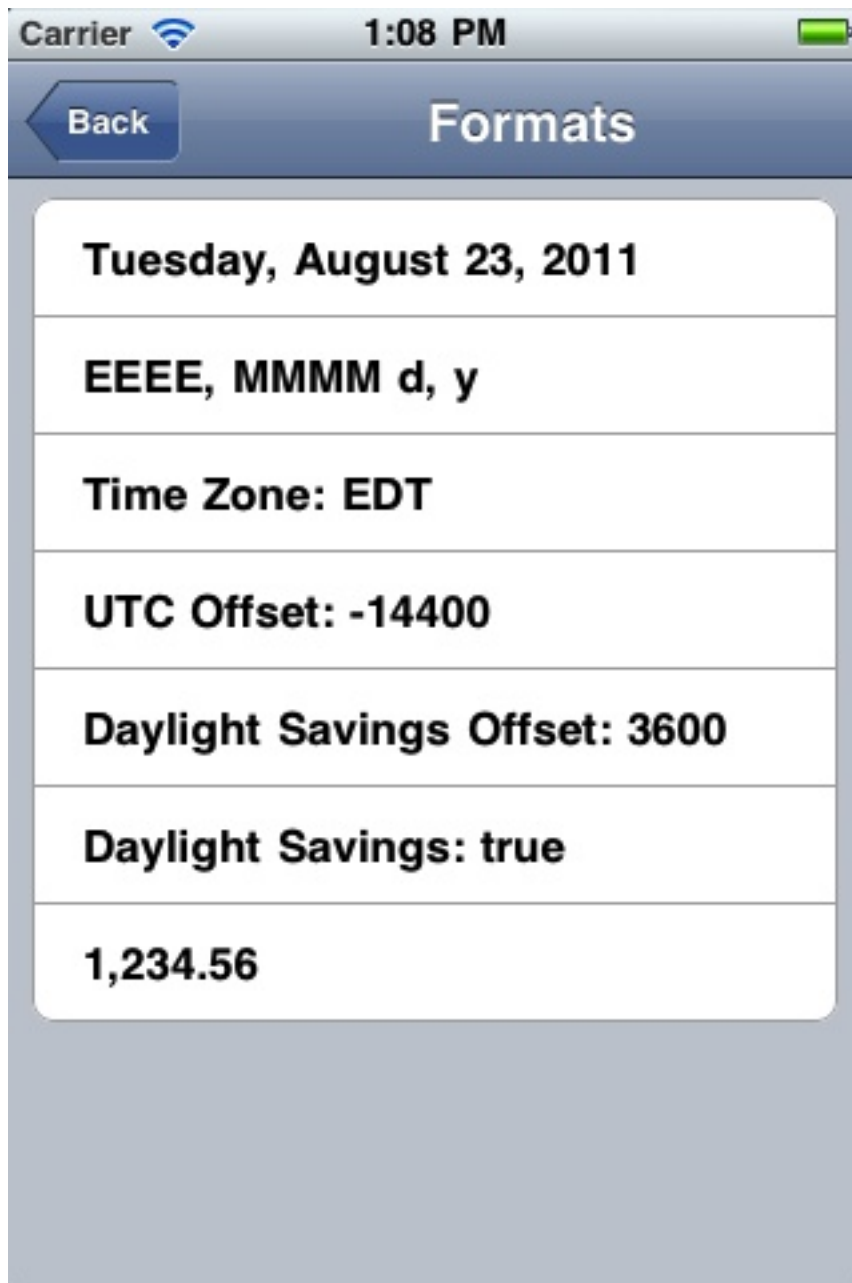
Figure 122. Using Cordova for cultural formatting

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 Foundation. Use 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 Foundation.

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 1005.

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 Foundation, you can globalize push notifications so that push
notifications are displayed in the language of the user. You use different methods
to globalize push notifications, depending on the way the application runs: in the
foreground, in the background, or not running at all.

Mobile applications frequently rely on server-side services to provide data to the
mobile application. However, there are occasions when the application is not
running or is not connected to the server. Push notifications are short messages
that provide a mechanism for notifying mobile applications that a server has data

that can be downloaded or information that can be viewed by the mobile application when the application is either not running or not running in the foreground.

Translate push notification messages so that the correct language is displayed to the user. How the application runs, such as, in the foreground, in the background, or not running at all, determines your choice of architectural pattern.

- When the application is running in the foreground, it uses the language and cultural settings on the device to determine the appropriate language to display. To support this pattern, messages must be stored in the resource files of the mobile application, and not in the resource files of the server application, even though messages are generated on the server-side.
- When a notification is sent to a mobile application, send the notification resource key and not the actual text of the message.
- When a mobile application receives the notification, or message, use the key that was sent in the notification message to look up the text of the message from its resource file, as shown in Figure 1.

Note: iOS uses Apple Push Notification Service (APNS) to push notifications to mobile applications. Android uses Google Cloud Messaging (GCM) to push notifications to mobile applications.
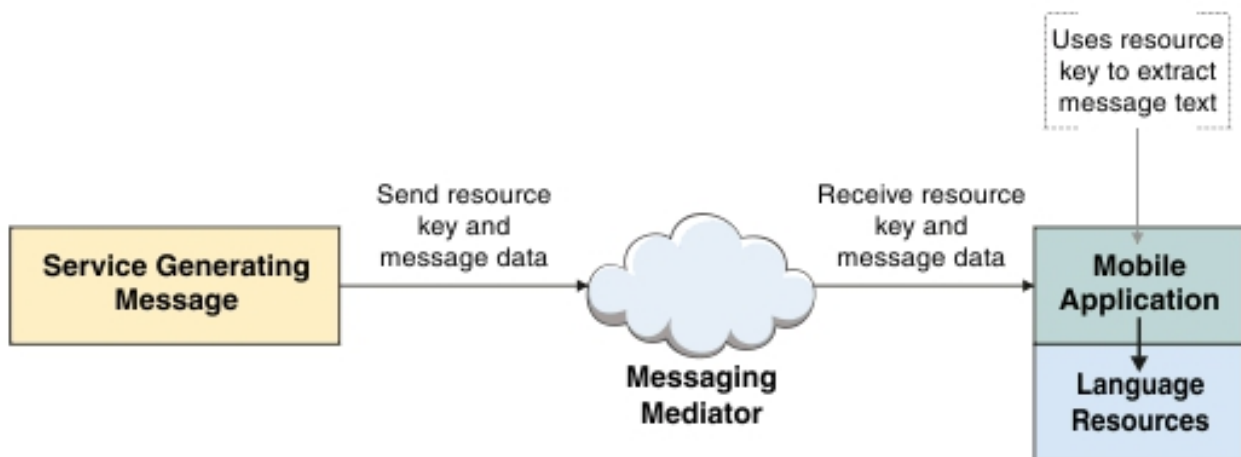
Figure 123. Using the resource key

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 a Worklight adapter to send the notification. For more information about how to create an adapter, see the first three modules about push notifications under category 8, *Notifications*, in Chapter 4, "Tutorials and samples," on page 35.

Listing 1: Send notification using created adapter

This listing shows how to send a notification with the created adapter. The target message, or resource key, to be translated on the client-side, is specified in the payload.

```
function sendNotificationOTA(userId, notificationText) {
  var userSubscription = WL.Server.getUserNotificationSubscription(
    'mysuranceAdapter.mysuranceEventSource', userId);
```

```
          WL.Server.notifyAllDevices(
            userSubscription,
            {
              badge : 1,
              sound : "",
              alert : notificationText,
              payload : { globalizeString : 'notificationText' }
            }
          );
}
```

Listing 2: Client-side subscription code

This listing shows the code that is required on the client-side to subscribe to push notification.

```
WL.Client.Push.onReadyToSubscribe = function(){
  WL.Client.Push.registerEventSourceCallback(
    "mysurancePush",
    "mysuranceAdapter",
    "mysuranceEventSource",
    pushNotificationReceived);
};
```

After successful subscription, the callback method is implemented. The callback method is responsible for retrieving the data from the payload, retrieving application locale preferences, retrieving the message by using the resource key, and formatting the message.

Listing 3: Callback method

This listing shows how to retrieve the locale information and load the corresponding translated message with Dojo by using the resource key that is stored in the payload object.

```
function pushNotificationReceived(props, payload){
  if (payload.globalizeString != "undefined"){
    require(
      ["dojox/mobile/i18n", "dojo/number"],
      function(mi18n, number){
        bundle = mi18n.load("resource", "messages");
        // get globalization text by dojo mobile i18n
        var notificationText = bundle[payload.globalizeString];
        // format number by device locale
        var num = number.format(1234567890, {
          places: 2, locale: WL.App.getDeviceLocale()});
        num = bundle["amount"] + num;
        //display globalization message
        alert(notificationText + "\n" +num);
      }
    );
  }
}
```

- If a notification provides data in addition to the message, then send the data in a locale-neutral format. When the application retrieves the message, the data can be formatted based on the user cultural preferences at the time the message is received.
- An application that is running in the background, or not running at all, can elect to use a previously registered user profile to access the appropriate language and cultural settings for push notifications. To support this pattern, the server sends the translated message and data in a format that is determined by the user cultural and language preferences that are stored in the profile, as shown in

Figure 2. The push notifications are then processed differently by the mobile application. Processing is based on the native operating system that the application is running on.
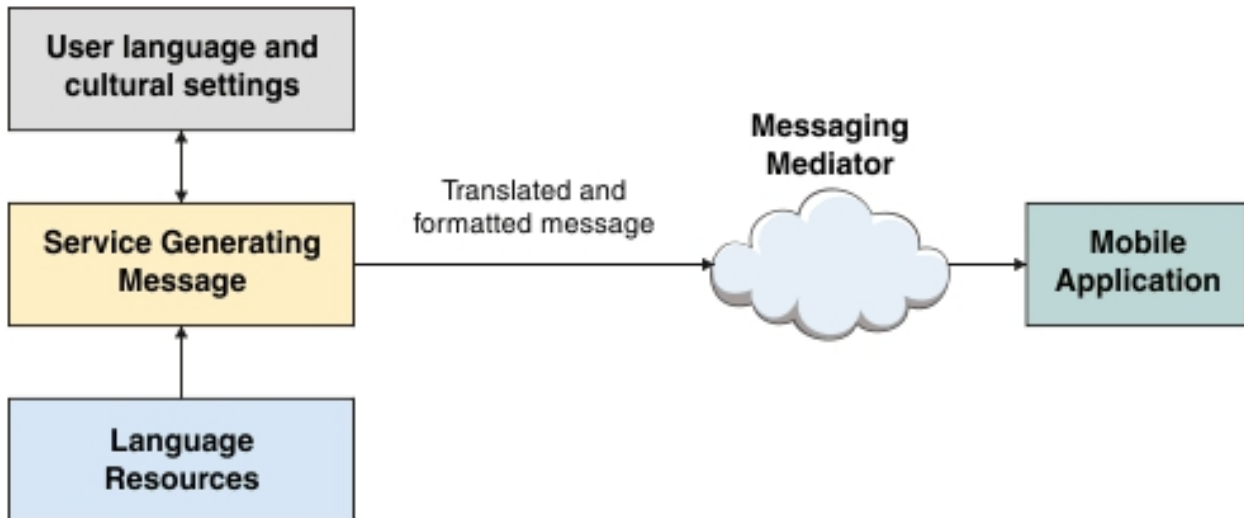
Figure 124. Sending push notifications according to the user's settings

- 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 Foundation, 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.

Enforce language preference for Worklight messages

IBM Worklight Foundation supports several languages, and based on the language of the device, appropriate translations are used to display system messages. However, you can enforce restriction to display messages only in a certain set of languages.

If your application supports only English and French, you can enforce a restriction so that Worklight displays the messages only in these languages.

Enforcing languages in hybrid and mobile web applications

You can provide a preferred list of languages to be used by Worklight for translating system messages in the `application-descriptor.xml` file. The order of the languages is important and must be separated by comma.

```
<languagePreferences>en, fr, de, es</languagePreferences>
```

Enforcing languages in native applications

You can provide a preferred list of languages in native environments by adding `languagePreferences` property in the appropriate client property file.

Fallback mechanism

The system compares the device locale with the values that are provided in the preference list. Here are the fallback mechanisms on how the system messages are displayed:

1. If the device locale is available, the corresponding translation is used to display system messages.
2. If the device locale (for example, `fr-FR`) is not in the preference list, or the locale is not supported by Worklight, the device language (for example, `fr`) is used to compare with the values that are provided in preference list. If the language is available in the preference list, the corresponding translation is used to display system messages.
3. If the device language (`fr`) is not in the preference list, or the language is not supported by Worklight, the first language available in the list for translation is used.

Note: The order of the languages that are specified in the preference list is important. You must list the order from generic to specific languages so that when the translation is not available, the generic language is loaded.

For the following example of the preference list, if the device language is set to `German`, Worklight uses English to display the system messages. The reason is that de is not available in the list.

```
<languagePreferences>en, fr, es</languagePreferences>
```

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 Foundation 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 Foundation 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 Foundation 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 Foundation, 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:
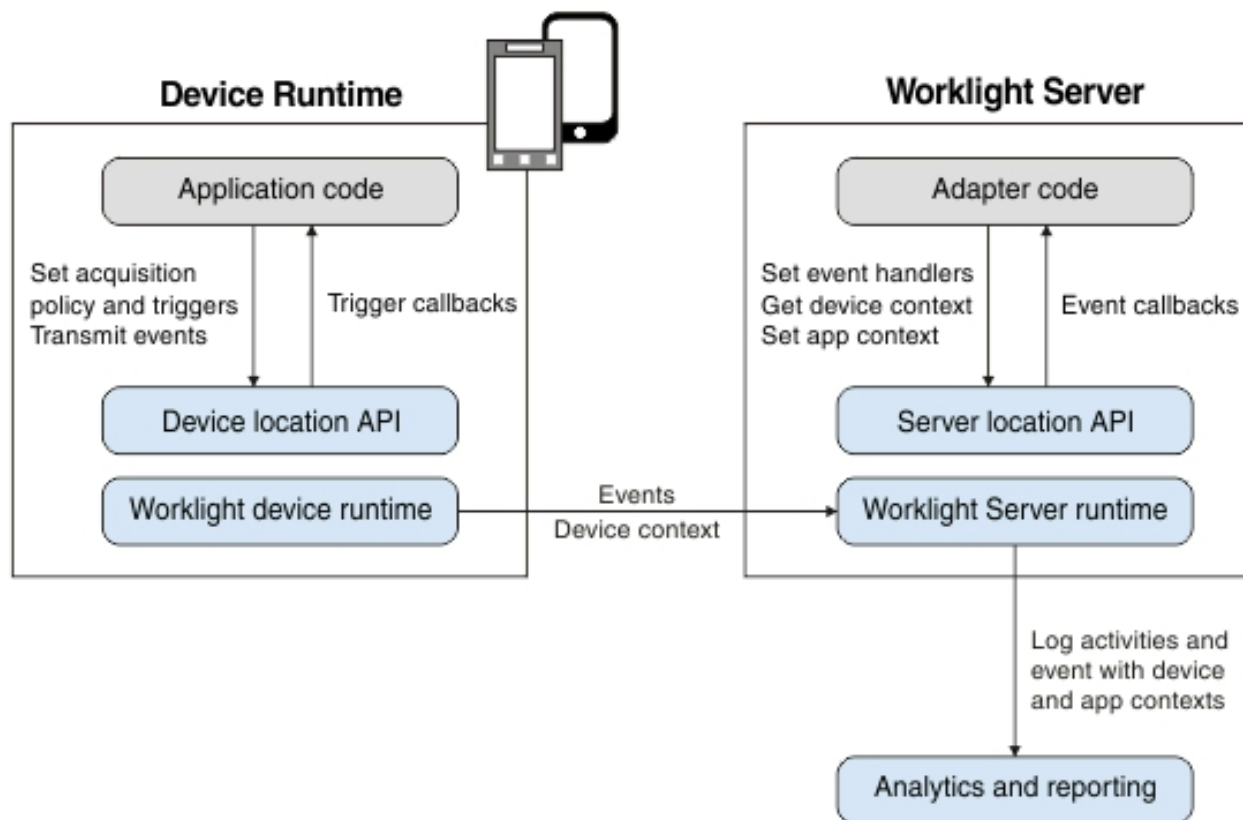
Figure 125. Location services architecture

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.

Related concepts:

"Testing hybrid location service applications" on page 539
You can use the Mobile Browser Simulator 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 Mobile Browser Simulator that use the JavaScript location service APIs.

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 Worklight location services, you must define the correct permissions.

Location services are supported for hybrid applications on Android, iOS, and Windows Phone 8.

Location services permissions in Android

The permissions that you require differ for versions earlier than Android 6.0 Marshmallow and versions starting from Android 6.0 Marshmallow onwards.

Note: Using Android 6.0 Marshmallow requires the latest interim fix of Worklight.

Before Android 6.0 Marshmallow

For Geo acquisition:
- ACCESS_COARSE_LOCATION
- ACCESS_FINE_LOCATION (when **enableHighAccuracy**=true)

For WiFi acquisition:
- ACCESS_WIFI_STATE
- CHANGE_WIFI_STATE

For Android 6.0 Marshmallow and later

In Android 6.0 Marshmallow and later, geo acquisition (location) permissions require additional runtime permissions.

Overview of runtime permissions

> According to the Android 6.0 Marshmallow permissions model, in addition to defining permissions at installation, users must also allow or deny access to different features at runtime. Before an app accesses location services, it must check whether permission has already been granted and, if needed, request permission. Developers are responsible to perform the check before accessing any of the following methods in the WLDevice interface:
>
> - startAcquisition
> - acquireGeoPosition
> - stopAcquisition
>
> If permissions have not been requested or not granted, the Worklight API does not get a provider and returns an error, WLGeoErrorCodes.PERMISSION_DENIED, along with a message about the requested accuracy level.

Checking and requesting permissions from the Android 6.0 Marshmallow API

> **Checking if permissions have been granted**
>> Two levels of access permission are available from android.Manifest.permission:
>>
>> - android.Manifest.permission.ACCESS_FINE_LOCATION
>> - android.Manifest.permission.ACCESS_COARSE_LOCATION
>>
>> In the examples that follow and in the sample that is provided, these methods are inherited by the MainActivity from the android.app.Activity class. To check the status of the permissions, call checkSelfPermission with the appropriate access level, as follows:
>>
>> `getContext().checkSelfPermission(android.Manifest.permission.ACCESS_FINE_LOCATION)`

> If permission has already been granted, the method returns the
> value PackageManager.PERMISSION_GRANTED.

Requesting permissions

> To ask the user for permission for the appropriate level of access,
> use the inherited requestPermissions method, as follows:

```
requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 0);
```

> Calling this Android method displays a dialog box that prompts
> the app to grant permission and invokes
> onRequestPermissionsResult when the choice is made. Once this
> permission has been granted to the system, startAcquisition,
> stopAcquisition, and acquireGeoPosition are granted access to the
> location services. If the user denies access, these methods return
> the WLGeoErrorCodes.PERMISSION_DENIED error. See the sample at
> https://developer.ibm.com/mobilefirstplatform/documentation/
> getting-started-7-1/foundation/advanced-client-side-development/
> location-services-hybrid-applications/ for a full demonstration of
> how to handle the new Android 6.0 Marshmallow permissions
> model.

Using Android location services in a hybrid Android application environment

> Android 6.0 Marshmallow requires user permissions to be granted at
> runtime. Because runtime JavaScript code in the web framework of the
> hybrid app does not have access to the Android API, these permissions
> must be requested and checked by the native code before launching
> Apache Cordova. The file *<application name>*.java in the native folder is
> responsible for loading the web resources. The code for requesting and
> checking permissions for location services can be called within the
> onInitWebFrameworkComplete API or within any other startup API in the
> class.

```
public void onInitWebFrameworkComplete(WLInitWebFrameworkResult result){
    if (result.getStatusCode() == WLInitWebFrameworkResult.SUCCESS) {
        super.loadUrl(WL.getInstance().getMainHtmlFilePath());
  } else {
      handleWebFrameworkInitFailure(result);
}

if (!(WLClient.getInstance().getContext().checkSelfPermission
    (android.Manifest.permission.ACCESS_FINE_LOCATION)== PackageManager.PERMISSION_GRANTED))
{
requestPermissions(new String[]{android.Manifest.permission.ACCESS_FINE_LOCATION}, 0);

    }
}
```

> This code requests permission if the permission has not already been
> granted. If the permission is still not granted, all requests for location
> services from the web framework fail.

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 959. 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 964.

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(){
 /*
  * Use of WL.Client.connect() API before any connectivity to a Worklight Server is required.
  * This API should be called only once, before any other WL.Client methods that communicate with the
  * Don't forget to specify and implement onSuccess and onFailure callback functions for WL.Client.co
  *
  *    WL.Client.connect({
  *      onSuccess: onConnectSuccess,
  *      onFailure: onConnectFailure
  *    });
  *
  */

 // Common initialization code goes here

 WL.App.hideSplashScreen();

}

  // 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:-73.958244323
                    [[WLCoordinate alloc] initWithLatitude:40.79691751000037 longitude:-73.949489593
                    [[WLCoordinate alloc] initWithLatitude:40.764486356929645 longitude:-73.97309303
                    [[WLCoordinate alloc] initWithLatitude:40.76799670467469 longitude:-73.981676101
                    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(){
 /*
  * Use of WL.Client.connect() API before any connectivity to a Worklight Server is required.
  * This API should be called once, before any other WL.Client methods that communicate with the W
  * Do not forget to specify and implement the onSuccess and onFailure callback functions for WL.C
  *
  *     WL.Client.connect({
  *        onSuccess: onConnectSuccess,
  *        onFailure: onConnectFailure
  *     });
  *
  */

 // Common initialization code goes here.

 WL.App.hideSplashScreen();

}

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 WLAcquisitionFailureCallbacksConfigurati
  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", "setAppCon
    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 = [

  -(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 = [[WLAcquisitionFailureCallbacksConfigura
        [failures setWifiFailureCallback: [WLCallbackFactory createWifiFailureCallback:^(WLWifiError* w
          [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_W
      [invocData setParameters:[NSArray arrayWithObject:[NSString stringWithFormat:@"{regionName: '%@'

      // Replace this code with a WLDelegate instance that will update the UI indicating success/failu
      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 database in any
}

// 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", MA
```

```
            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 required 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 required
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, required 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.

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. 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.

Starting in IBM Worklight Foundation V6.2.0, developers that use Worklight client-side APIs who want to capture both platform (IBM Worklight Foundation) and application (your code) logs for debug and problem determination should use the appropriate client-side APIs. By doing so, debug log data is made available for capture and sending to the server.

Introduction to client-side logging

The APIs that are available with Worklight client libraries include a logger in JavaScript, Android native, and iOS native code base. The logger API is similar to commonly used logger APIs, such as `console.log` (JavaScript), `java.util.logging`

(Java), and `nslog` (Objective-C). The Worklight logger API has the additional capability of persistently capturing logged data for sending to the server to be used for analytics gathering and developer inspection. Use the logger APIs to report log data at appropriate levels so that developers who inspect logs can triage and fix problems without having to reproduce problems in their labs.

There are seven levels. From least verbose to most verbose, they are FATAL, ERROR, WARN, INFO, LOG, DEBUG, TRACE.

Example usage of level-appropriate messages:
- Use TRACE for method entry and exit points.
- Use DEBUG for method result output.
- Use LOG for class instantiation.
- Use INFO for initialization reporting.
- Use WARN to log deprecated usage warnings.
- Use ERROR for unexpected exceptions or unexpected network protocol errors.
- Use FATAL for unrecoverable crashes or hangs.

Default log capture feature behavior
- Log capture is ON.
- During development, the default log level is DEBUG.
  - On iOS, development mode means that the DEBUG macros is defined.
  - On Android, development mode means that the app is signed with the default Android certificate.
- In production, the default log level is FATAL.
  - On iOS, production mode means that the DEBUG macros is not defined.
  - On Android, production mode means that the app is signed with the customer's generated certificate.
  - The FATAL level is reserved for applications that experience unrecoverable errors, which appear to users as an application crash or hang. The Worklight client-side library records unrecoverable errors when log capture is ON and logger is active.
- Log persistent client-side buffer maximum size is 100k bytes.
  - Log entries are treated as a first in, first out (FIFO) queue; oldest log entries are deleted to make room for more recent log entries.
- Client retrieves log configuration from the server after each successful client network `init` sequence and `invokeProcedure` response.
  - Turn this automatic behavior on or off by using one or more of the following options:
    - `Logger.setAutoUpdateConfig(boolean)`
    - `OCLogger.setAutoUpdateConfig(boolean)`
    - `WL.Logger.config({autoUpdateConfig: boolean})`
  - After automatic behavior is turned off, you must explicitly call the `updateConfigFromServer` method in your application to enable the use of the admin-controlled log configuration panel on the Worklight console.
- All captured log data, if any, is sent to the Worklight Server during each successful client network `init` sequence and `invokeProcedure` response, with a 60 second buffer.
  - Turn this automatic behavior on or off by using one or more of the following options:

- `Logger.setAutoSendLogs(boolean)`
- `OCLogger.setAutoSendLogs(boolean)`
- `WL.Logger.config({autoSendLogs: boolean})`

– After automatic behavior is turned off, you must explicitly call the `send` method (in both the Logger and Analytics classes) in your application to send any persistently captured client logs to the Worklight Server.

• (Android) Native Logger API is not active, and not capturing logs or data about unrecoverable errors, unless and until the `Logger.setContext(Context)` method is called.

– This API call is made automatically under the `WL.init()` method call.

During development

• Developers should make liberal and reasonable use of the client-side logger APIs.

• Client-side logs that are uploaded to the embedded Liberty server in Worklight Studio are written to files under the `clientlogs` folder. This folder is a peer to the `logs` folder of the embedded server.

– Verifying this behavior is a good way to confirm the expected behavior of your usage of the API.

In production

• Logger configuration is controllable from the Worklight Console. Configuration that is retrieved from the server is used as an override of the locally set configuration. Clients revert to the pre-override configuration when the Worklight administrator removes the logger configuration and the client retrieves the instruction from the server.

Things to consider

During application development, consider the following questions.

Should capture be always on or always off?
The default setting of capture is ON. When capture is on, all logs at the specified level or filter are captured in a persisted rotating log buffer. You can change the default of the capture setting by using the logger API.

Consider that turning capture on at a verbose logger level has an impact to resource consumption:

• CPU

• file system space

• frequency of network usage when captured log data is also being sent to the server

• size of network payload when captured log data is also being sent to the server

At what level should you set the logger?
There are seven levels. From least verbose to most verbose, they are FATAL, ERROR, WARN, INFO, LOG, DEBUG, TRACE.

For example, when capture is ON and the logger level is configured to FATAL, the logger captures uncaught exceptions. Uncaught exceptions often appear to users as application crashes or hangs, but does not capture any logs that lead up to the failure. Alternatively, a more verbose logger level ensures that logs that lead to a logger FATAL entry are captured.

Consider that verbose logger levels, when capture is ON, can affect:
- frequency of network usage
- size of the payload that is sent to the server
- application performance, and therefore user experience

How frequently should clients check with the server for logger configuration changes?

By default, client applications check for updated logger configuration during the Worklight client network `init` sequence, which is not necessarily application startup or application foreground events.

The `init` sequence can be infrequent, depending on the design of your application. For example, the `init` sequence might happen only at check-out in a retail shopping application. In this example, the application can check for new configuration on every `onForeground` event to ensure that it retrieves and applies the configuration soon after the Worklight administrator sets in the Catalog tab of the Worklight Console.

For example, to retrieve and apply configuration overrides from the server when the client comes to the foreground, you can place the WLClient `updateConfigFromServer` function call:
- `onResume` (Android, in each Activity, if necessary)
- `applicationDidBecomeActive` (iOS)
- an `onForeground` event listener (JavaScript)

How can you guarantee that all captured log data on the client gets to the server?

The short answer is that there is no way to guarantee preservation of all captured data. Clients might be running the application offline and simultaneously accumulating captured log data. Because the client is offline with limited file system space, older log data must be purged in favor of preserving more recent log data, which is the behavior of the log capture feature.

You can make a best effort at ensuring that all captured data gets to the server by applying one or more of the following strategies:
- Call the `send` function on a time interval.
- Trigger a call to the `send` function on application lifecycle events, like pause and resume events.
- Batch the `send` call with other application network activity, like `invokeProcedure`. This approach allows the device radio to sleep and preserve battery.
- Increase the capacity of the persistent log buffer on the client by calling the `setMaxFileSize` function.

How can you capture logs from your application only, and exclude logger entries from Worklight code?

If your application code is making good use of the Worklight logger API, and you want to capture logs from your application only, you can use a consistent package name or consistent set of package names for your logger instances. For example:
- `// JavaScript`
  `var logger = WL.Logger.create('MyAppPkg');`
- `// Android`
  `Logger logger = new Logger('MyAppPkg');`
- `// iOS`
  `OCLogger* logger = [OCLogger getInstanceWithPackage:@"MyAppPkg"];`

Then, set the filters on the logger to allow logging only for your package or packages:
- // JavaScript
  WL.Logger.setFilters({'MyAppPkg':'debug'});
- // Android
  HashMap filters = new HashMap();
  filters.put("MyAppPkg", Logger.LEVEL.DEBUG);
  Logger.setFilters(filters);
- // iOS
  [OCLogger setFilters:@{@"MyAppPkg": @(OCLogger_DEBUG)}];

How can you never collect or send logs from deployed apps in the field?

Call setCapture(false) as early as possible in your application lifecycle code to set the default behavior. Avoid the logger tab in the Catalog tab of the Worklight Console.

Server preparation for uploaded log data

You must prepare your server to receive uploaded client log data.

Upon receiving uploaded client logs, the Worklight production server passes the uploaded data to the Operational Analytics component feature and to an adapter that you create and deploy. Neither of these options are present in a production Worklight Server; you must install and configure them. To receive and persist uploaded client logs at the Worklight Server, you must take at least one of the following two actions:

1. Install the IBM Worklight Analytics Platform as described in "Installing the IBM Worklight Analytics Platform" on page 232.
2. Deploy an adapter that is named WLClientLogReceiver or the name that corresponds to the value of the wl.clientlogs.adapter.name JNDI property.

If you deploy an adapter to receive uploaded client logs, the adapter must be an HTTP adapter that is named WLClientLogReceiver or the value of the wl.clientlogs.adapter.name JNDI property. The adapter must have at least one procedure that 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 762.

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" />
```

The implementation of the adapter determines the destination of the uploaded log content.

One convenient way to persistently record uploaded client logs is to place the `audit="true"` attribute in the adapter's procedure element. This flag instructs the Worklight Server to report all adapter invocations and parameter arguments inline to the server log file:

```
<procedure name="log" audit="true" />
```

Alternatively, you process the parameters that are passed into the `log` procedure explicitly.

Server security

By default, the servlet that receives uploaded client logs at the Worklight Server is configured in the `authenticationConfig.xml` file with a security test that includes application authenticity check. You can change the security tests that protect the servlet. But to avoid unexpectedly prompting the user for authentication credentials when you send logs, you have two choices:

1. Use a security test that requires no custom challenge handler code and no user interaction, and freely call the logger `send` function.
2. Ensure that the security test in front of the servlet remains the same as the security test of the application, and be careful about placement of extra logger `send` function calls.

If you choose to change the security test, and you choose option one, an explicit call to the logger `send` function does not result in an unexpected authentication challenge prompt or other authentication failure. The logger `send` function is safe to place throughout your application.

If you choose to change the security test, and you choose option two, a carelessly placed call to the logger `send` function might result in an unexpected authentication challenge prompt or other authentication failure. In this case, explicit calls to the logger `send` function in your application must be placed carefully. If your client applications call the logger `send` function explicitly, ensure that they do so after authentication succeeds. For example, call the logger `send` function in the `invokeProcedure onSuccess` callback of an adapter invocation that is protected by the same security test as the log receiver servlet.

Logging sensitive data

The logger library does not automatically protect against logging sensitive data. Data is stored in plain text, but is only readable within the context of the application that is using the logger API. Avoid logging sensitive data

Uploaded client logs

In the embedded Liberty development server in Worklight Studio, uploaded client logs are written to a file that corresponds with that client's unique attributes. The uploaded client log file is written, or appended, at the following path, which is a peer to the `logs` folder:

`clientlogs/[os]/[os_version]/[app_id]/[app_version]/[device_id].log`

Uploaded logs are not written to the file system in Worklight production servers.

Client-side log capture configuration from the Worklight Console

Starting in IBM Worklight Foundation V6.2, administrators can use the Log Configuration subtab in the Catalog tab of the Worklight Console to affect client logger configuration. Administrators can adjust the log level and log package filters for any combination of operating system, operating system version, application, application version, and device model.

When the Worklight administrator creates a configuration profile, it takes effect at the corresponding client application upon the next client application network `init` sequence or upon the next explicit WLClient `updateConfigFromServer` function call, whichever comes first.

When the Worklight administrator removes a configuration profile, the client reverts to its configuration before the server configuration profile override upon the next client application `init` sequence or upon the next explicit WLClient `updateConfigFromServer` function call, whichever comes first.

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.
>
> Alternatively, developers can use standard `java.util.logging.Logger` in their code with the understanding that levels, filtering, and capturing are still controlled in the `com.worklight.common.Logger` class. No `java.util.logging.Logger` method calls are captured until the `com.worklight.common.Logger.setContext(Context)` method is called.

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 iOS apps" on page 1005.

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 API.

Chapter 10. Testing with IBM Worklight Foundation

The mobile testing features of IBM Mobile Test Workbench for Worklight automate the creation, execution, and analysis of functional tests for Worklight native, hybrid, and web applications on Android and iOS devices. A superset of these features is available in Rational Test Workbench Mobile Test Edition. Testing BlackBerry applications is not currently supported.

Installation

To use the test workbench, you must install it as an extra component in Worklight Studio. For instructions about installing the test workbench within Worklight Studio, see "Installing and configuring IBM Mobile Test Workbench for Worklight" on page 68.

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.

Tools for testing mobile applications in IBM Worklight Foundation

The tools for preparing and testing mobile applications in IBM Worklight Foundation include:
- 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.
- A mobile test client that runs on Android and iOS devices, as well as Android emulators and iOS simulators. This client is used to record, to run tests, and to view reports. With the Android client, you can also upload apps to the test workbench.

  The Android client is a native, Android application. The iOS client is a Web app that runs in a browser on the iOS device. In addition, there is a native iOS client that can be used to test with the iOS Simulator. For details, see Android testing overview and iOS testing overview.
- The test workbench, which works together with the mobile test client for testing the application. See more details below.

The test workbench

The test workbench runs on a Windows, Linux, or Macintosh computer and includes the following main components:
- The Test Navigator lists test projects, tests, mobile devices, and the mobile incoming recordings that are used to generate tests.

- The Mobile Devices editor lists the devices that are connected to the test workbench. This editor displays detailed specifications of each device, which allows you to select the hardware platforms on which you can deploy and run your tests.

- The Mobile and Web UI Applications 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, stubs, and so on.

- A Mobile Data view displays the screen captures that were uploaded from the mobile device during the recording. Use this view to display and select user interface elements, and optionally to add verification points to the test script.

Stages in the testing process

The goal of mobile testing is to ensure that your mobile application meets the requirements that guided its design and development. To help you meet this goal, IBM Mobile Test Workbench for Worklight implements the following stages in the testing process:

- **Configuration**: Set up your test environment with IBM Mobile Test Workbench for Worklight and the Android SDKs for the mobile operating systems. Install the mobile test client on one or more Android devices. Ensure that the mobile devices have connectivity through WiFi, 3G, or 4G, and add those devices to the test workbench. For details, see Configuring the mobile test client.
- **Application preparation**: Import the application that you want to test into the test workbench, or for Android applications, use the device to upload the application under test to the test workbench. For iOS applications, instrument and install the applications. For details, see "Managing mobile applications" on page 992.
- **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. For details, see "Creating mobile tests" on page 992.
- **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. For details, see "Editing mobile tests" on page 993.
- **Test execution**: 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. For details, see "Running mobile tests" on page 993.
- **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. For details, see "Evaluating results" on page 993.

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 than 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 additional stages:

- **Publication**: You can publish Android applications to the Application Center from Worklight Studio by right-clicking an Android project and clicking **IBM Application Center** > **Publish on IBM Application Center**. For iOS, the application must first be instrumented for testing. You right-click an iOS project, and click **IBM Application Center** > **Publish Test-Ready Application**. The iOS application is then instrumented and published.
- **Import of the application**: When the application is published in the Application Center, the tester imports this app into the list of managed applications into the test workbench.

Support for testing native, hybrid, and web applications

Use the test workbench to test various types of mobile applications, including native applications, hybrid applications, and browser-based, web 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.

A browser-based *web* application is developed using pure web technologies, such as HTML 5, CSS3, and JavaScript libraries, such as Dojo and JQuery. Web applications are developed to run in multiple browsers and are platform-independent. This release includes support for Dojo Mobile 1.9 and jQuery Mobile 1.3.

Note: To test applications that are not created with IBM Worklight Foundation, you must use IBM Rational Test Workbench or IBM Rational Test Workbench Mobile Test Edition.

Compound tests

If you need to combine various mobile tests into a single workflow or end-to-end scenario, you can organize the tests into a compound test. Each test may perform a part of the scenario. For details, see Compound tests.

Note: Rational Test Workbench is required to combine mobile tests with other types of tests in a compound test.

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. For details, see Extending test execution with custom code.

Using IBM Rational Test Workbench with IBM Worklight Foundation

You can enhance the testing capabilities in IBM Mobile Test Workbench for Worklight by licensing either IBM Rational Test Workbench Mobile Test Edition or the full IBM Rational Test Workbench product.

With both products, you can test mobile apps that are developed with tools other than IBM Worklight Foundation.

The full IBM Rational Test Workbench product provides several extra capabilities:
- Integration with IBM Rational Quality Manager (RQM), which provides advanced test execution and test management capabilities. When you provide IBM Rational Test Workbench access to the test assets in the project created with IBM Mobile Test Workbench for Worklight, you can use RQM to drive test execution. For more information, see the IBM Rational Quality Manager section in IBM Rational solution for Collaborative Lifecycle Management.
- Ability to test non-mobile applications, such as desktop Web UI applications, Selenium, HTTP, Citrix, SAP, and other test domains. In addition, you can combine multiple types of tests in a single compound test and run them as a single workflow.
- Load testing with IBM Rational Performance Tester
- Test virtualization with IBM Rational Integration Tester

Getting started

For conceptual information about testing Android and iOS applications, as well as flow diagrams and step-by-step instructions, see the topics and tutorials listed below:
- Getting started with mobile testing
- Videos about mobile testing
- Tutorial: Test a native Android app
- Tutorial: Test a native iOS app

Creating a Test Workbench project

The tests that you create, and the assets associated with the tests, reside in a Test Workbench project. You can create a Test Workbench project in the test workbench itself, or you can create one when you create your Worklight project. This topic describes the steps for creating the Test Workbench project when you create your Worklight project.

About this task

For details about creating a Test Workbench project in the test workbench itself, see Creating a Test Workbench project.

Note: The IBM Mobile Test Workbench for Worklight allows testing only the mobile applications that are created with IBM Worklight Foundation. To test applications that are not created with IBM Worklight Foundation, 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 a Worklight project (see "Creating Worklight projects" on page 414).
2. On the last page of the project creation wizard, click **IBM Mobile Test Workbench**, select **Create a Test Project** and enter the name of the test project.
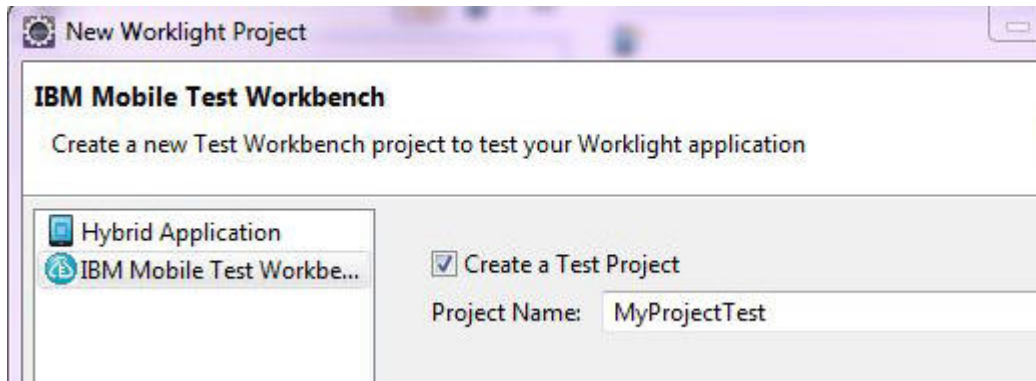
Figure 126. Creating a Test Project from the wizard

> 3. Click **Finish**.

Managing mobile applications

Before you can record a test from an Android or iOS application, the application must be instrumented, imported into the test workbench, and installed on a mobile device. Android applications can be instrumented by uploading the application from a mobile device to the test workbench, or by importing the application into the test workbench. iOS applications are instrumented by running a script on a Macintosh computer.

You can view the mobile applications that are available for testing, as well as information about each application in the Mobile and Web UI Applications editor, as shown below:

For details about managing mobile applications for testing, see the following topics and video:
- Importing applications to test in the workbench
- Uploading Android applications from the mobile test client
- Instrumenting iOS applications
- Installing instrumented iOS applications
- Tutorial: Test a native Android app, Preparing app for recording
- Tutorial: Test a native iOS app, Preparing the application under test
- Instrumenting and installing iOS apps in Rational Test Workbench v8.6 (video)

Creating mobile tests

You create a mobile test by recording a session with the app under test. At the end of the session, the recording is uploaded to the test workbench, where it is used to generate a test.

For details about recording mobile tests, see the following topics, tutorials, and video:
- Recording tests from the Android mobile test client
- Recording tests from the iOS mobile test client
- Tutorial: Test a native Android app, Recording a test

- Tutorial: Test a native iOS app, Recording a test
- Recording a Mobile App Test with Rational Test Workbench (video)

Editing mobile tests

After you record a test, you can edit the test in the natural language editor, which allows you to modify the test manually.

The edited test displays the list of actions and UI elements uploaded from a mobile device during the recording. The upper left view of the test editor, Test Contents, displays the chronological sequence of events in the test. The view on the right, User Action Details, displays details about the currently selected action in the test script.

You can use the Mobile Data view at the bottom to display and select UI elements from the recorded application. You can replace recorded test values with variable test data, or add dynamic data to the test.

For details about editing mobile tests, see the following topics, tutorials, and video:
- Editing mobile tests
- Actions from the Mobile and Web UI data view
- Tutorial: Test a native Android app, Editing a test
- Tutorial: Test a native iOS app, Creating a verification point
- Tutorial: Test a native iOS app, Enhancing a test with a loop and a datapool
- Rational Test Workbench - Mobile technical preview overview

Running mobile tests

You can run tests from mobile devices, Android emulators, and iOS Simulators. You can also initiate a test run from the test workbench.

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.

For details about running mobile tests, see the following topics and video:
- Running mobile tests
- Running tests from an Android mobile test client
- Running tests from an iOS mobile test client
- Tutorial: Test a native iOS app, Running a test
- Running Mobile App Tests with Rational Test Workbench (video)

Evaluating results

You evaluate test results by viewing the report that is generated at the end of a test run. Functional reports provide a comprehensive view of the behavior of the app under test. Reports can be exported and archived for validation.

For details about viewing test results and test logs, see the following topics and video:
* Evaluating results
* Managing logs for the Android mobile test client
* Test log overview
* Tutorial: Test a native Android app, Evaluating functional results
* Tutorial: Test a native iOS app, Running the test and evaluating the results
* Running Mobile App Tests with Rational Test Workbench (video)

Using Worklight Studio and Application Center

These topics describe how to initiate tests from Worklight Studio and how to work with Application Center.

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 Worklight hybrid applications. To test native applications that you created with a 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 Worklight application.
2. Click **Run As** > **Test with IBM Mobile Test Workbench**.
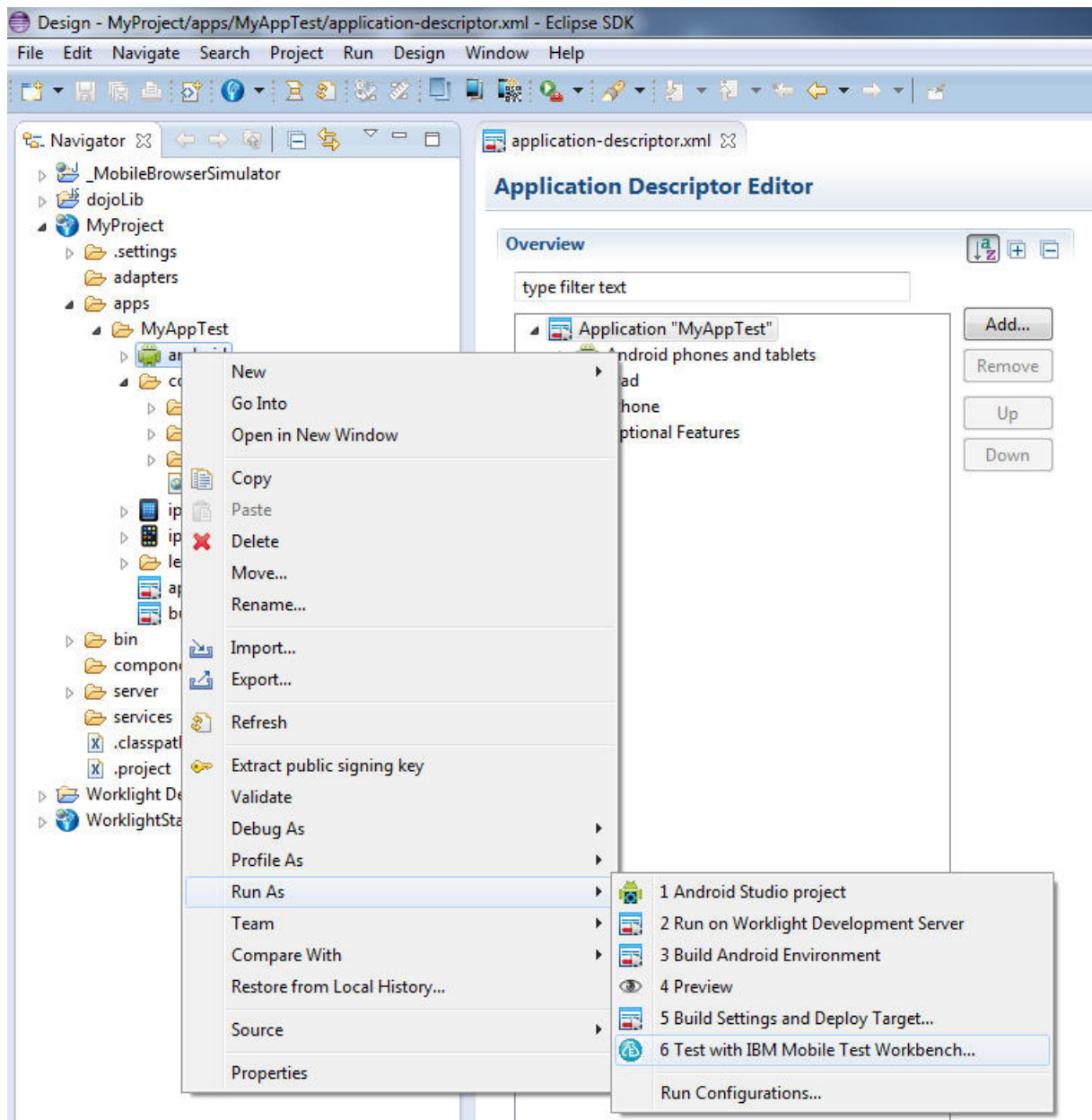
Figure 127. Testing with IBM Mobile Test Workbench from Worklight Studio

- 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 Worklight Android environment in your application. See "Publishing Worklight applications to the Application Center" on page 1347 for more information.

To publish the APK file to the Application Center, choose **IBM Application Center** > **Publish on IBM Application Center**.

No specific instrumentation is required for Android applications. You can use any Android application that is available in the Application Center for testing.

Sharing iOS applications for testing

Testing iOs applications requires that each application is instrumented before you can record tests on it with IBM Mobile Test Workbench for Worklight. Applications must be instrumented before they are published in the Application Center catalog.

To instrument and publish an iOS application to the Application Center in a single operation, choose **IBM Application Center** > **Publish test-ready application**. This operation is only available when you run Worklight Studio on MacOS.

The instrumentation of the application uses the Xcode project to do the instrumentation. See Publishing test-ready iOS applications to the Application Center for more information.

Alternatively, you can use a script to instrument an iOS application. See Command line to launch the rtwBuildXcode.sh script for the command line and parameters to use to instrument an iOS application manually. This script produces an archive file that you can manually upload to the Application Center console.

An instrumented iOS application appears with a special test icon in the Application Center catalog. The icon for IBM Mobile Test Workbench for Worklight overlays the application icon.
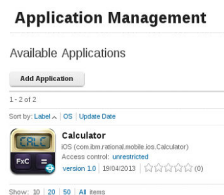
Figure 128. Instrumented application in the Application Center console

Importing applications into the mobile test workbench

When an application is published for test purposes in the Application Center, a tester can import the application into the list of managed applications in IBM Mobile Test Workbench for Worklight.

In the Perspectives toolbar of IBM Mobile Test Workbench for Worklight, click this icon ▤ to open the editor for mobile applications. In this editor, you can browse the applications available for testing in the Application Center. You can select the applications that you want to import into IBM Mobile Test Workbench for Worklight. See Adding applications in the workbench for more information.

Publishing test-ready iOS applications to the Application Center

Deploy iOS applications that are ready to be tested with the Mobile Test Workbench to the Application Center directly from the Worklight Studio IDE.

About this task

Worklight Studio provides an easy way to publish test-ready iOS applications to the Application Center. In Worklight Studio, you can instrument a Worklight application for mobile testing and publish it to the Application Center. When an

application is available in the Application Center, a member of another team can easily import it into the Mobile Test Workbench for testing.

Procedure

1. Specify the publication preferences for the Application Center.
   a. In the main menu, click **Window** > **Preferences**.
   b. In the tree on the left, expand IBM Application Center and select **Publish Preferences**.
   c. Enter the user credentials and server URL for publishing a Worklight application to the Application Center

      See this table for a description of the required publication preferences.
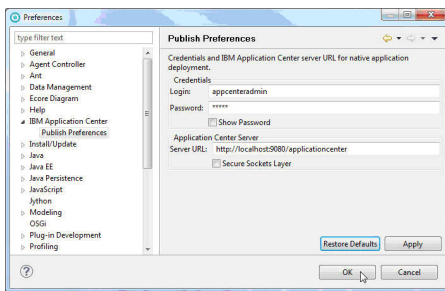
Figure 129. User credentials and server URL for deploying applications to the Application Center

Table 157. Publication preferences for deploying an application to the Application Center

Preference	Description
Credentials	Login name and password for accessing the repository of the Application Center.
Server URL	URL of the Application Center server to use for publishing applications.

2. Publish an iOS application to the Application Center.
   a. Right-click the iPad or iPhone environment of the IBM Worklight project, or the Xcode project directory, and select **IBM Application Center** > **Publish Test-Ready Application**. The instrumentation of the project starts.
   b. When instrumentation is complete, click **Publish** to publish the application with the current preferences or click **Preferences** to change any of the preferences before publishing.
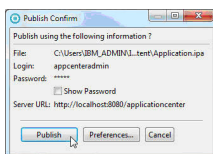
Figure 130. Options to publish the application or change the publication preferences

If the application already exists, publication will fail.
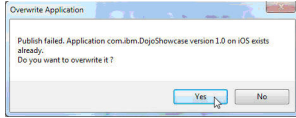
Figure 131. Failed publication of an existing published application

    c. **Option for existing published applications**: Select **Yes** to overwrite the existing version of the application and to publish the new version.

Chapter 11. API reference

To develop your native or hybrid applications, refer to the Worklight API in JavaScript, Java Platform, Micro Edition (Java ME), Java for Android, and Objective-C for iOS.

Use Worklight API to develop your applications in JavaScript, Java Platform, Micro Edition (Java ME), Java for Android, and Objective-C for iOS.

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 Foundation.

You can use Worklight client-side API capabilities to improve application development, and Worklight server-side API to improve client/server integration and communication between mobile applications and back-end systems.

With the Worklight client-side API, your mobile application has access to various 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.

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. 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 Worklight client-side API provide access to 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 Foundation.

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 Worklight technology, and Worklight Server. 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 Foundation 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 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 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 Foundation 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 158. 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 158. 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 158. 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 Foundation. 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 iOS apps

You can use Objective-C API to develop apps for the iOS environment.

Use the Objective-C client-side API for iOS apps that IBM Worklight Foundation provides if you want to access Worklight services from iOS applications.

You can use this API to develop native apps. If you develop hybrid apps, you can also use the relevant part of this API, either directly by using the WL.NativePage API or by using Cordova.

For more information about this API, expand the entry for this topic in the **Contents** panel and see the **Overview** topic listed there.

Java client-side API for Android apps

You can use Java API to develop apps for the Android environment.

Use the Java client-side API for Android apps that IBM Worklight Foundation provides if you want to access Worklight services from Android mobile applications.

You can use this API to develop native apps. If you develop hybrid apps, you can also use the relevant part of this API, either directly by using the WL.NativePage API or by using Cordova.

For more information about this API, expand the entry for this topic in the **Contents** panel and see the **Overview** topic listed there.

Java client-side API for Java Platform, Micro Edition (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 Foundation provides if you want to access Worklight services from Java ME apps.

For more information about this API, expand the entry for this topic in the **Contents** panel and see the **Overview** topic listed there.

C# client-side API for Windows Phone 8 apps

You can use C# API to develop apps for the Windows Phone 8 environment.

Use the C# client-side API for Windows Phone 8 apps that IBM Worklight Foundation provides if you want to access Worklight services from Windows Phone 8 mobile applications.

You can use this API to develop native apps. If you develop hybrid apps, you can also use the relevant part of this API, either directly by using the WL.NativePage API or by using Cordova.

You can find the description of this API in the following document: C# client-side API for Windows Phone 8 apps

Worklight server-side API

Use the server-side API that IBM Worklight Foundation 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 Foundation 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 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 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 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 Worklight database tables, their description, and how they are used.

Name	Description	Order of Magnitude
CLUSTER_SYNC	Internal cluster synchronization tasks.	10s of rows.
CONFIG_PROFILES	Stores configuration profiles that are added to the server through the console.	1 row per profile.
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_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.
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**.
PUSH_DEVICES	Push notification table. Stores a record per device.	1 row per device.
PUSH_SUBSCRIPTIONS	Push notification table. Stores a record per tag subscription or user subscription to event sources.	1 row per device subscription.
OPENJPA_SEQUENCE_TABLE	Internal table created for JPA. Not used, and will 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.
WORKLIGHT_VERSION	The Worklight version.	1 row.

The following table provides a list of common 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 1424. | The size depends on application. The customer is responsible for purging older entries after aggregating to Data Warehouse. |
| 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 1428. | 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. |

Name	Description	Order of Magnitude
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.

The following table provides a list of common Worklight Administration database tables, their description, and how they are used.

Name	Description	Order of Magnitude
ADAPTERS	Stores the adapter deployable elements. This table is used to synchronize the adapter deployable elements between cluster nodes. Many-to-many relationships with the PROJECTS table.	10s of rows.
APPLICATIONS	Stores the application deployable elements. This table is used to synchronize the application deployable elements between cluster nodes. Many-to-many relationships with the PROJECTS table.	10s of rows.
APP_VERSION_ACCESS_DATA	Stores the applications that have the remote disable mode to block or notify. References the APPLICATIONS_ENVIRONMENTS table.	10s of rows.
APPLICATIONS_ENVIRONMENTS	Environments (for example, iPhone, Android) of deployed applications. Many-to-one relationships with the APPLICATIONS table.	10s of rows.
AUDIT_TRAIL	Stores an audit trail of all administrative actions performed on the administration server.	1,000s of rows.
PROJECT	Stores the names of the deployed projects.	10s of rows.

Name	Description	Order of Magnitude
PROJECT_ADAPTERS	Bidirectional association between projects and adapters.	10s of rows.
PROJECT_APPLICATIONS	Bidirectional association between projects and applications.	10s of rows.
PROJECT_LOCK	Internal cluster synchronization tasks.	10s of rows.
PUSH_ENVIRONMENTS	Push notification table. Stores details of push environments.	10s of rows.
PUSH_TAGS	Push notification table. Stores details of tags defined.	10s of rows.
TRANSACTIONS	Internal cluster synchronization table storing the state of all current administrative actions.	10s of rows.
WORKLIGHTMGT_VERSION	The IBM Worklight Foundation product version.	1 row.

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 159. 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 160. 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 Worklight Server	

Table 160. Application API request elements (continued)

Header Name	Data Type	Description	Valid values
Application ID	String	Unique Identifier of the application within the Worklight Server. Every application deployed on the Worklight Server must have a unique identifier	Up to 256 alphanumeric and underscore characters
Application Environment	String	Name of the environment that 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 161. 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.

Table 161. Actions (continued)

Action	HTTP Request	Parameters
logout	POST	**x, isAjaxRequest** - see the following table showing common parameters.
login	POST	**x, isAjaxRequest** – see the following table showing common parameters. **realm** – string.
updates	POST	**x, isAjaxRequest** – see the following table showing common parameters. **skin** – current skin name (string) **checksum** – the checksum of the current skin (string) **skinLoaderChecksum** – the checksum of the skin selection code (string)
getup	POST	**x, isAjaxRequest** - see the following table showing common parameters.
deleteup	POST	**x, isAjaxRequest** – see the following table showing common parameters. **userprefkey** – the user preference to delete.
getuserinfo	POST	**x, isAjaxRequest** – see the following table showing common parameters.
getgadgetprefs	POST	**x, isAjaxRequest** - see the following table showing common parameters.

Table 161. Actions (continued)

Action	HTTP Request	Parameters
`notifications`	POST	`x, isAjaxRequest` – see the following table showing common parameters. `subscribe` – `JSON` string containing subscribe options `unsubscribe` – when specified, designates an unsubscribe action `updateToken` – the update notification token (string) `adapter` – the name of the notification adapter (string) `eventSource` – the name of the notification event source (string) `alias` – notification subscription alias (string) `tag` – the name of the tag (string)
`fbcallback`	GET or POST	`x, isAjaxRequest` – see the following table showing common parameters. `popup` – string
`composite`	POST	`x, isAjaxRequest` - see the following table showing common parameters. `requests` – a `JSON` string containing information about other actions to invoke. This action is used to combine several actions in a single `HTTP` request.
`appversionaccess`	GET	`x, isAjaxRequest` – see the following table showing common parameters.
`setup`	POST	`x, isAjaxRequest` - see the following table showing common parameters. `userprefs` contains JSON pairs of preference key and value

Table 161. Actions (continued)

Action	HTTP Request	Parameters
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 162. 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 submit a web application resource request:

`{Protocol}://{Worklight Server}/apps/services/www/{Application ID}/{Application Environment}/{Appl`

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **Application ID**, and **Application Environment**.

Table 163. Request elements

Element	Data Type	Description	Valid Values
Application Resource Path	String	HTML, image, JavaScript, CSS, and any other application resource	Example values: img/bg.png, myWidget.html, js/myWidget.js

Preview application resource requests

Use the following request structure to preview application resource requests:

```
{Protocol}://{Worklight Server}/apps/services/preview/{Application ID}/{Application Environment}/{App
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **ApplicationID**, and **Application Environment**.

Table 164. Request elements

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`

REST Services API

The REST API provides several services to administer the runtime environments concerning adapters, applications, devices, audit, transactions, security, and push notifications.

The REST service API for adapters and applications for each runtime environment is located in `/management-apis/1.0/runtimes/`*runtime-name*`/`, where *runtime-name* is the name of the runtime environment that is administered through the REST service. Then, the type of object addressed by the service is identified together with the appropriate method. For example, `/management-apis/1.0/runtimes/`*runtime-name*`/Adapters` (`POST`) refers to the service for deploying an adapter.

Adapter Binary (GET, HEAD)

Retrieves the binary of a specific adapter.

Description

It supports range requests to deliver only a range of the bytes of the adapter. Clients can use this feature to resume a download after interruption.

Roles

Users in the following roles are authorized to perform this operation:
- `worklightadmin`
- `worklightdeployer`

Method

`GET`, `HEAD`

Path

/management-apis/1.0/runtimes/*runtime-name*/downloads/adapters/*adapter-name*

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/downloads/adapters/
```

Path Parameters

runtime-name
  The name of the runtime. This is the contex root of the runtime web
  application, without the leading slash.

adapter-name
  The name of the adapter.

Query Parameters

Query parameters are optional.

locale
  The locale used for error messages.

Produces

application/octet-stream

Response

The binary data of the specified adapter.

Errors

400
```
The request is invalid.
```

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime or the adapter is not found.
```

416
```
The requested range of bytes is not satisfiable.
```

500
```
An internal error occurred.
```

Adapter (DELETE)

Deletes a specific adapter.

Description

This transaction can run synchronously or asynchronously. If processed
asynchronously, the REST service returns before the transaction is completed. In
this case, you can query the transaction result later with the transaction REST
service.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/adapters/*adapter-name*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/myadapter?asy

Path Parameters

runtime-name
 The name of the runtime. This is the contex root of the runtime web
 application, without the leading slash.

adapter-name
 The name of the adapter.

Query Parameters

Query parameters are optional.

async
 Whether the transaction is processed synchronously or asynchronously.
 Allowed values are true and false. The default is synchronous processing.

locale
 The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deleted adapter.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "filename" : "myadapter.adapter",
      "name" : "myadapter",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
```

```
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "DELETE_ADAPTER",
    "userName" : "demouser",
  },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-adapter-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_ADAPTER"
    userName="demouser">
    <description
      filename="myadapter.adapter"
      name="myadapter"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-adapter-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
   The exact product version.

transaction
   The details of the transaction.

The *transaction* has the following properties:

appServerId
   The id of the web application server.

description
   The details of the adapter.

errors
   The errors occurred during the transaction.

id The id of the transaction.

project
   The current project.

status
    The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
    The date in ISO 8601 format when the adapter was created.

timeUpdated
    The date in ISO 8601 format when the adapter was updated.

type
    The type of the transaction, here always DELETE_ADAPTER.

userName
    The user that initiated the transaction.

The *description* has the following properties:

filename
    The optional file name of the adapter.

name
    The name of the adapter.

The *error* has the following properties:

details
    The main error message.

The *project* has the following properties:

name
    The name of the project, which is the context root of the runtime.

Errors

403

The user is not authorized to call this service.

404

The corresponding runtime or the adapter is not found.

500

An internal error occurred.

Adapter (GET)

Retrieves meta information of a specific adapter.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/adapters/*adapter-name*

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/myadapter?
```

Path Parameters

runtime-name
    The name of the runtime. This is the contex root of the runtime web
    application, without the leading slash.

adapter-name
    The name of the adapter.

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the specified adapter.

Example as JSON

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "description" : "My first sample adapter",
  "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapter
  "name" : "myadapter",
  "platformVersion" : "6.1.0.00.20131126-0630",
  "procedures" : [ "getSomething", ... ],
  "productVersion" : "6.2.0",
  "projects" : [
    {
      "name" : "myproject",
    },
    ...
  ],
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter
  deployTime="2014-04-13T00:18:36.979Z"
  description="My first sample adapter"
  link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/my
  name="myadapter"
  platformVersion="6.1.0.00.20131126-0630"
```

```
    productVersion="6.2.0">
    <procedures>
      <procedure>getSomething</procedure>
      ...
    </procedures>
    <projects>
      <project name="myproject"/>
      ...
    </projects>
</adapter>
```

Response Properties

The response has the following properties:

deployTime
>   The date in ISO 8601 format when the adapter was deployed.

description
>   The description of the adapter.

link
>   The URL to access detail information about the adapter.

name
>   The name of the adapter.

platformVersion
>   The exact version number of the Worklight development tools (Studio) that
>   built the adapter.

procedures
>   The JavaScript procedures of the adapter.

productVersion
>   The exact product version.

projects
>   The projects the adapter belong to.

The *project* has the following properties:

name
>   The name of the project, which is the context root of the runtime.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or the adapter is not found.`

500

`An internal error occurred.`

Adapter (POST)

Deploys an adapter.

Description

It first checks whether the input adapter is valid. Then, it transfers the adapter to the data base and to the runtime.

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

POST

Path

/management-apis/1.0/runtimes/*runtime-name*/adapters

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters?async=fals

Path Parameters

runtime-name
   The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

async
   Whether the transaction is processed synchronously or asynchronously. Allowed values are true and false. The default is synchronous processing.

locale
   The locale used for error messages.

Consumes

multipart/form-data

Produces

application/json, application/xml, text/xml

Response

The meta data of the deployed adapter.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "alreadyDeployed" : false,
      "filename" : "myadapter.adapter",
      "name" : "myadapter",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "UPLOAD_ADAPTER",
    "userName" : "demouser",
  },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-adapter-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_ADAPTER"
    userName="demouser">
    <description
      alreadyDeployed="false"
      filename="myadapter.adapter"
      name="myadapter"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</deploy-adapter-result>
```

Response Properties

The response has the following properties:

ok   Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the adapter.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction, here always UPLOAD_ADAPTER.

userName
> The user that initiated the transaction.

The *description* has the following properties:

alreadyDeployed
> Whether a version of the adapter was already previously deployed.

filename
> The optional file name of the adapter.

name
> The name of the adapter.

The *error* has the following properties:

details
> The main error message.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

403

The user is not authorized to call this service.

404

The corresponding runtime is not found.

500

```
An internal error occurred.
```

Adapters (GET)

Retrieves meta information for the list of deployed adapters.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
GET
```

Path

/management-apis/1.0/runtimes/*runtime-name*/adapters

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters?locale=de_DE&
```

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
> The locale used for error messages.

offset
> The offset from the beginning of the list if only a part of the list (a page) should be returned.

orderBy
> The sort mode. By default, the elements are sorted in increasing order. If the sort mode starts with - (minus sign), the elements are sorted in decreasing order. Possible sort modes are: name, deployTime. The default sort mode is: name.

pageSize
> The number of elements if only a part of the list (a page) should be returned. The default value is 100.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deployed adapters.

Example as JSON

```
{
  "items" : [
    {
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "description" : "My first sample adapter",
      "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/ada
      "name" : "myadapter",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "procedures" : [ "getSomething", ... ],
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
    },
    ...
  ],
  "pageSize" : 100,
  "productVersion" : "6.2.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<adapters
  pageSize="100"
  productVersion="6.2.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deployTime="2014-04-13T00:18:36.979Z"
      description="My first sample adapter"
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapter
      name="myadapter"
      platformVersion="6.1.0.00.20131126-0630">
      <procedures>
        <procedure>getSomething</procedure>
        ...
      </procedures>
      <projects>
        <project name="myproject"/>
        ...
      </projects>
    </item>
    ...
  </items>
</adapters>
```

Response Properties

The response has the following properties:

items
> The array of adapter meta information

pageSize
The page size if only a page of adapters is returned.

productVersion
The exact product version.

startIndex
The start index in the total list if only a page of adapters is returned.

totalListSize
The total number of adapters.

The *adapter* has the following properties:

deployTime
The date in ISO 8601 format when the adapter was deployed.

description
The description of the adapter.

link
The URL to access detail information about the adapter.

name
The name of the adapter.

platformVersion
The exact version number of the Worklight development tools (Studio) that built the adapter.

procedures
The JavaScript procedures of the adapter.

projects
The projects the adapter belong to.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

403
The user is not authorized to call this service.

404
The corresponding runtime is not found.

500
An internal error occurred.

App Version Access Rule (PUT)

Sets the access rule of a specific app version.

Description

The access rule specifies the behavior when a user accesses the application on the device.

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*/ *application-env*/*application-version*/accessRule

Example

`https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappl`

Path Parameters

runtime-name
  The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

application-name
  The name of the application.

application-env
  The application environment.

application-version
  The application version number.

Query Parameters

Query parameters are optional.

async
  Whether the transaction is processed synchronously or asynchronously. Allowed values are `true` and `false`. The default is synchronous processing.

locale
  The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

Example as JSON

```
{
  "action" : "NOTIFY",
  "downloadLink" : "ibmappctr://myapp",
  "message" : "Please update!",
  "multiLanguageMessage" : [
    {
      "locale" : "de",
      "message" : "Bitte updaten!",
    },
    ...
  ],
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<accessrule
  action="NOTIFY"
  downloadLink="ibmappctr://myapp"
  message="Please update!">
  <multiLanguageMessage>
    <localizedMessage
      locale="de"
      message="Bitte updaten!"/>
    ...
  </multiLanguageMessage>
</accessrule>
```

Payload Properties

The payload has the following properties:

action
> The action to be performed. It can have the following values: NOTIFY (notify the user of some message), BLOCK (block the execution the application), NA (nothing is performed), DELETE (remove the access rule).

downloadLink
> An optional link displayed with the message where to download a new version of the application.

message
> The message to be displayed when the action is NOTIFY or BLOCK.

multiLanguageMessage
> Messages in additional languages

The *multilanguage message* has the following properties:

locale
> The locale of the message.

message
> The translated message.

Response

The meta data of the app version and its access rule.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "action" : "NOTIFY",
      "appVersion" : {
        "applicationName" : "myapplication",
        "environment" : "android",
        "version" : "1.0",
      },
      "createdAtDate" : "2014-02-13T00:18:36.979Z",
      "message" : "This version is no longer supported.",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "SET_APPLICATION_ENV_VERSION_ACCESS_RULE",
    "userName" : "demouser",
  },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-accessrule-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="SET_APPLICATION_ENV_VERSION_ACCESS_RULE"
    userName="demouser">
    <description
      action="NOTIFY"
      createdAtDate="2014-02-13T00:18:36.979Z"
      message="This version is no longer supported.">
      <appVersion
        applicationName="myapplication"
        environment="android"
        version="1.0"/>
    </description>
    <errors>
      <error details="An internal error occured."/>
      ...
```

```
        </errors>
        <project name="myproject"/>
    </transaction>
</set-appversion-accessrule-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the application.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING,
> SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state
> SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction, here always
> SET_APPLICATION_ENV_VERSION_ACCESS_RULE.

userName
> The user that initiated the transaction.

The *description* has the following properties:

action
> The action to be done when a disabled app version is accessed. Possible values
> are: NOTIFY, BLOCK, NA, DELETE.

appVersion
> The corresponding app version

createdAtDate
> The date in ISO 8601 format when the app version access rule was created.

downloadLink
> The download link where to obtain a new version of the application.

message
The message to be displayed when a disabled app version is accessed.

The *app version* has the following properties:

applicationName
The name of the application.

environment
The environment of the application.

version
The version of the application.

The *error* has the following properties:

details
The main error message.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

The payload is invalid.

403

The user is not authorized to call this service.

404

The corresponding runtime or the app version is not found.

500

An internal error occurred.

App Version Authenticity Check (PUT)

Sets the authenticity check rule of a specific app version.

Description

This transaction can run synchronously or asynchronously. If processed
asynchronously, the REST service returns before the transaction is completed. In
this case, you can query the transaction result later with the transaction REST
service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*/
application-env/*application-version*/applicationAuthenticity

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myapplica

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web
> application, without the leading slash.

application-name
> The name of the application.

application-env
> The application environment.

application-version
> The application version number.

Query Parameters

Query parameters are optional.

async
> Whether the transaction is processed synchronously or asynchronously.
> Allowed values are true and false. The default is synchronous processing.

locale
> The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

Example as JSON

```
{
  "action" : "DISABLED",
}
```

Payload Properties

The payload has the following properties:

action
> The action to check the authenticity. It can have the following values: DISABLED

(authenticity is not checked - all clients pass authenticity check), `ENABLED` (authenticity is always checked - clients need to respond as expected), `IGNORED` (authenticity is checked but ignored - all clients pass but warnings are issued to server log).

Response

The meta data of the app version and its authenticity check rule.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "appVersion" : {
        "applicationName" : "myapplication",
        "environment" : "android",
        "version" : "1.0",
      },
      "newAuthValue" : "DISABLED",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "SET_APPLICATION_ENV_AUTHENTICITY_CHECK_RULE",
    "userName" : "demouser",
  },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-authenticitycheckrule-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="SET_APPLICATION_ENV_AUTHENTICITY_CHECK_RULE"
    userName="demouser">
    <description newAuthValue="DISABLED">
      <appVersion
        applicationName="myapplication"
        environment="android"
        version="1.0"/>
    </description>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
```

```
      <project name="myproject"/>
  </transaction>
</set-appversion-authenticitycheckrule-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
>   The exact product version.

transaction
>   The details of the transaction.

The *transaction* has the following properties:

appServerId
>   The id of the web application server.

description
>   The details of the application.

errors
>   The errors occurred during the transaction.

id The id of the transaction.

project
>   The current project.

status
>   The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING,
>   SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state
>   SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
>   The date in ISO 8601 format when the adapter was created.

timeUpdated
>   The date in ISO 8601 format when the adapter was updated.

type
>   The type of the transaction, here always
>   SET_APPLICATION_ENV_AUTHENTICITY_CHECK_RULE.

userName
>   The user that initiated the transaction.

The *description* has the following properties:

appVersion
>   The corresponding app version

newAuthValue
>   The new authenication rule (DISABLED, ENABLED, IGNORED).

The *app version* has the following properties:

applicationName
>   The name of the application.

environment
>   The environment of the application.

version
    The version of the application.

The *error* has the following properties:

details
    The main error message.

The *project* has the following properties:

name
    The name of the project, which is the context root of the runtime.

Errors

400

```
The payload is invalid.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or the app version is not found.
```

500

```
An internal error occurred.
```

App Version (DELETE)

Deletes a specific app version.

Description

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*/
application-env/*application-version*

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappl
```

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

application-name
> The name of the application.

application-env
> The application environment.

application-version
> The application version number.

Query Parameters

Query parameters are optional.

async
> Whether the transaction is processed synchronously or asynchronously. Allowed values are true and false. The default is synchronous processing.

locale
> The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deleted app version.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "applicationName" : "myapplication",
      "environment" : "android",
      "version" : "1.0",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "DELETE_APPLICATION_ENV_VERSION",
    "userName" : "demouser",
  },
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<delete-appversion-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_APPLICATION_ENV_VERSION"
    userName="demouser">
    <description
      applicationName="myapplication"
      environment="android"
      version="1.0"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-appversion-result>
```

Response Properties

The response has the following properties:

ok   Whether the transaction was successful.

productVersion
>   The exact product version.

transaction
>   The details of the transaction.

The *transaction* has the following properties:

appServerId
>   The id of the web application server.

description
>   The details of the application.

errors
>   The errors occurred during the transaction.

id   The id of the transaction.

project
>   The current project.

status
>   The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
>   The date in ISO 8601 format when the adapter was created.

timeUpdated
>   The date in ISO 8601 format when the adapter was updated.

type
>   The type of the transaction, here always DELETE_APPLICATION_ENV_VERSION.

userName
>   The user that initiated the transaction.

The *description* has the following properties:

applicationName
>   The name of the application.

environment
>   The environment of the application.

version
>   The version of the application.

The *error* has the following properties:

details
>   The main error message.

The *project* has the following properties:

name
>   The name of the project, which is the context root of the runtime.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or the app version is not found.
```

500

```
An internal error occurred.
```

App Version Lock (PUT)

Locks a specific app version.

Description

A locked app version cannot be updated anymore.

Roles

Users in the following roles are authorized to perform this operation:
* **worklightadmin**
* **worklightdeployer**
* **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*/
application-env/*application-version*/lock

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappl
```

Path Parameters

runtime-name
The name of the runtime. This is the contex root of the runtime web
application, without the leading slash.

application-name
The name of the application.

application-env
The application environment.

application-version
The application version number.

Query Parameters

Query parameters are optional.

locale
The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

Example as JSON

```
{
  "lock" : true,
}
```

Payload Properties

The payload has the following properties:

lock
Whether the app version is locked.

Response

Example as JSON

```
{
  "ok" : true,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-lock-result ok="true"/>
```

Response Properties

The response has the following properties:

ok Whether the operation was successful.

Errors

400

```
The payload is invalid.
```

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or the app version is not found.
```

500

```
An internal error occurred.
```

Application Binary (GET, HEAD)

Retrieves the binary of a specific app version.

Description

It supports range requests to deliver only a range of the bytes of the app version.
Clients can use this feature to resume a download after interruption.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

```
GET, HEAD
```

Path

/management-apis/1.0/runtimes/*runtime-name*/downloads/applications/
application-name/application-env/application-version

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/downloads/application
```

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web
> application, without the leading slash.

application-name
The name of the application.

application-env
The application environment.

application-version
The application version number.

Query Parameters

Query parameters are optional.

locale
The locale used for error messages.

Produces

application/octet-stream

Response

The binary data of the specified app version.

Errors

400
```
The request is invalid.
```

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime or the app version is not found.
```

416
```
The requested range of bytes is not satisfiable.
```

500
```
An internal error occurred.
```

Application (DELETE)

Deletes a specific application and all its app versions.

Description

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myapplic

Path Parameters

runtime-name
 The name of the runtime. This is the contex root of the runtime web
 application, without the leading slash.

application-name
 The name of the application.

Query Parameters

Query parameters are optional.

async
 Whether the transaction is processed synchronously or asynchronously.
 Allowed values are true and false. The default is synchronous processing.

locale
 The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deleted application.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "applicationName" : "myapplication",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
```

```
      "type" : "DELETE_APPLICATION",
      "userName" : "demouser",
    },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-application-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="DELETE_APPLICATION"
    userName="demouser">
    <description applicationName="myapplication"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</delete-application-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the application.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction, here always DELETE_APPLICATION.

userName
> The user that initiated the transaction.

The *description* has the following properties:

applicationName
> The name of the application.

The *error* has the following properties:

details
> The main error message.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

403

The user is not authorized to call this service.

404

The corresponding runtime or the application is not found.

500

An internal error occurred.

Application (GET)

Retrieves meta information of a specific application.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/applications/*application-name*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myapplica

Path Parameters

`runtime-name`
    The name of the runtime. This is the contex root of the runtime web
    application, without the leading slash.

`application-name`
    The name of the application.

Query Parameters

Query parameters are optional.

`locale`
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the specified application.

Example as JSON

```
{
  "description" : "My first sample application",
  "displayName" : "My Sample Application",
  "environments" : [
    {
      "applicationEnvironmentDataAccess" : {
        "action" : "NOTIFY",
        "createdTime" : "2014-04-13T00:18:36.979Z",
        "message" : "This version is no longer supported.",
      },
      "authenticityCheckRule" : "DISABLED",
      "buildTime" : "2014-03-29T00:18:36.979Z",
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "deviceProvisioningRealm" : "myProvRealm",
      "environment" : "android",
      "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/app
      "securityTest" : "mobileTest",
      "supportRemoteDisable" : true,
      "supportsAuthenticity" : true,
      "userAuthenticationRealm" : "myAuthRealm",
      "version" : "1.0",
      "versionLocked" : false,
    },
    ...
  ],
  "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
  "name" : "myapplication",
  "platformVersion" : "6.1.0.00.20131126-0630",
  "productVersion" : "6.2.0",
  "projects" : [
    {
      "name" : "myproject",
    },
    ...
  ],
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<application
  description="My first sample application"
  displayName="My Sample Application"
  link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/r
  name="myapplication"
  platformVersion="6.1.0.00.20131126-0630"
  productVersion="6.2.0">
  <environments>
    <environment
      authenticityCheckRule="DISABLED"
      buildTime="2014-03-29T00:18:36.979Z"
      deployTime="2014-04-13T00:18:36.979Z"
      deviceProvisioningRealm="myProvRealm"
      environment="android"
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applicatio
      securityTest="mobileTest"
      supportRemoteDisable="true"
      supportsAuthenticity="true"
      userAuthenticationRealm="myAuthRealm"
      version="1.0"
      versionLocked="false">
      <applicationEnvironmentDataAccess
        action="NOTIFY"
        createdTime="2014-04-13T00:18:36.979Z"
        message="This version is no longer supported."/>
    </environment>
    ...
  </environments>
  <projects>
    <project name="myproject"/>
    ...
  </projects>
</application>
```

Response Properties

The response has the following properties:

description
　　The description of the application.

displayName
　　The display name of the application.

environments
　　The array of application environments.

link
　　The URL to access detail information about the application.

name
　　The name of the application.

platformVersion
　　The exact version number of the Worklight development tools (Studio) that
　　built the application.

productVersion
　　The exact product version.

projects
　　The projects the application belong to.

The *environment* has the following properties:

applicationEnvironmentDataAccess
The access rule to be executed when the app version is disabled.

authenticityCheckRule
Whether the authenticity is checked. Possible values are: ENABLED, IGNORED, DISABLED.

buildTime
The time stamp when the app version was built.

deployTime
The date in ISO 8601 format when the adapter was deployed.

deviceProvisioningRealm
The name of the realm used for device provisioning.

environment
The platform environment of the app version: android, iphone, ...

link
The URL to access detail information about the application version.

publishUrl
For web applications, this is the URL under which the web application was published.

securityTest
The name of the security test for a protected resource.

supportRemoteDisable
true if the application version supports remote disabling.

supportsAuthenticity
true if the application version supports authenticatuib.

userAuthenticationRealm
The name of the realm used to authenticate users.

version
The version number of the app version.

versionLocked
Whether the version is locked.

The *applicationEnvironmentDataAccess* has the following properties:

action
The action to be done when a disabled app version is accessed. Possible values are: NOTIFY, BLOCK, NA.

createdTime
The date in ISO 8601 format when the app version access rule was created.

downloadLink
The download link where to obtain a new version of the application.

message
The message to be displayed when a disabled app version is accessed.

multiLanguageMessage
Internationalized variants of the message to be displayed when a disabled app version is accessed.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or the application is not found.
```

500

```
An internal error occurred.
```

Application (POST)

Deploys an application.

Description

It first checks whether the input applicatio is valid. Then, it transfers the application to the data base and to the runtime.

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

POST

Path

/management-apis/1.0/runtimes/*runtime-name*/applications

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications?async=fa
```

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

async
> Whether the transaction is processed synchronously or asynchronously.
> Allowed values are `true` and `false`. The default is synchronous processing.

locale
> The locale used for error messages.

Consumes

multipart/form-data

Produces

application/json, application/xml, text/xml, text/html

Response

The meta data of the deployed application.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "appVersionsAlreadyDeployed" : [
        {
          "applicationName" : "myapplication",
          "environment" : "android",
          "version" : "1.0",
        },
        ...
      ],
      "appVersionsDeployed" : [
        {
          "applicationName" : "myapplication",
          "environment" : "android",
          "version" : "1.0",
        },
        ...
      ],
      "filename" : "myapplication.wlapp",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "UPLOAD_APPLICATION",
    "userName" : "demouser",
  },
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<deploy-application-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="UPLOAD_APPLICATION"
    userName="demouser">
    <description filename="myapplication.wlapp">
      <appVersionsAlreadyDeployedArray>
        <appVersionsAlreadyDeployed
          applicationName="myapplication"
          environment="android"
          version="1.0"/>
        ...
      </appVersionsAlreadyDeployedArray>
      <appVersionsDeployedArray>
        <appVersionsDeployed
          applicationName="myapplication"
          environment="android"
          version="1.0"/>
        ...
      </appVersionsDeployedArray>
    </description>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</deploy-application-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the application.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
The date in ISO 8601 format when the adapter was created.

timeUpdated
The date in ISO 8601 format when the adapter was updated.

type
The type of the transaction, here always UPLOAD_APPLICATION.

userName
The user that initiated the transaction.

The *description* has the following properties:

appVersionsAlreadyDeployed
The app versions that were already previously deployed and remain unchanged.

appVersionsDeployed
The app versions deployed.

filename
The optional file name of the application.

The *app version* has the following properties:

applicationName
The name of the application.

environment
The environment of the application.

version
The version of the application.

The *error* has the following properties:

details
The main error message.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

403
The user is not authorized to call this service.

404
The corresponding runtime is not found.

500
An internal error occurred.

Applications (GET)

Retrieves meta information for the list of deployed applications.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/applications

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications?locale=de

Path Parameters

runtime-name
  The name of the runtime. This is the contex root of the runtime web
  application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
  The locale used for error messages.

offset
  The offset from the beginning of the list if only a part of the list (a page)
  should be returned.

orderBy
  The sort mode. By default, the elements are sorted in increasing order. If the
  sort mode starts with - (minus sign), the elements are sorted in decreasing
  order. Possible sort modes are: name, deployTime. The default sort mode is:
  name.

pageSize
  The number of elements if only a part of the list (a page) should be returned.
  The default value is 100.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deployed applications.

Example as JSON

```
{
  "items" : [
    {
      "description" : "My first sample application",
      "displayName" : "My Sample Application",
      "environments" : [
        {
          "applicationEnvironmentDataAccess" : {
            "action" : "NOTIFY",
            "createdTime" : "2014-04-13T00:18:36.979Z",
            "message" : "This version is no longer supported.",
          },
          "authenticityCheckRule" : "DISABLED",
          "buildTime" : "2014-03-29T00:18:36.979Z",
          "deployTime" : "2014-04-13T00:18:36.979Z",
          "deviceProvisioningRealm" : "myProvRealm",
          "environment" : "android",
          "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime
          "securityTest" : "mobileTest",
          "supportRemoteDisable" : true,
          "supportsAuthenticity" : true,
          "userAuthenticationRealm" : "myAuthRealm",
          "version" : "1.0",
          "versionLocked" : false,
        },
        ...
      ],
      "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/app
      "name" : "myapplication",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
    },
    ...
  ],
  "pageSize" : 100,
  "productVersion" : "6.2.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<applications
  pageSize="100"
  productVersion="6.2.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      description="My first sample application"
      displayName="My Sample Application"
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
      name="myapplication"
      platformVersion="6.1.0.00.20131126-0630">
      <environments>
        <environment
          authenticityCheckRule="DISABLED"
          buildTime="2014-03-29T00:18:36.979Z"
          deployTime="2014-04-13T00:18:36.979Z"
          deviceProvisioningRealm="myProvRealm"
```

```
                environment="android"
                link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/appli
                securityTest="mobileTest"
                supportRemoteDisable="true"
                supportsAuthenticity="true"
                userAuthenticationRealm="myAuthRealm"
                version="1.0"
                versionLocked="false">
                <applicationEnvironmentDataAccess
                  action="NOTIFY"
                  createdTime="2014-04-13T00:18:36.979Z"
                  message="This version is no longer supported."/>
            </environment>
            ...
        </environments>
        <projects>
          <project name="myproject"/>
          ...
        </projects>
      </item>
      ...
    </items>
</applications>
```

Response Properties

The response has the following properties:

items
    The array of application meta information

pageSize
    The page size if only a page of applications is returned.

productVersion
    The exact product version.

startIndex
    The start index in the total list if only a page of applications is returned.

totalListSize
    The total number of applications.

The *application* has the following properties:

description
    The description of the application.

displayName
    The display name of the application.

environments
    The array of application environments.

link
    The URL to access detail information about the application.

name
    The name of the application.

platformVersion
    The exact version number of the Worklight development tools (Studio) that
    built the application.

projects
    The projects the application belong to.

The *environment* has the following properties:

applicationEnvironmentDataAccess
The access rule to be executed when the app version is disabled.

authenticityCheckRule
Whether the authenticity is checked. Possible values are: ENABLED, IGNORED, DISABLED.

buildTime
The time stamp when the app version was built.

deployTime
The date in ISO 8601 format when the adapter was deployed.

deviceProvisioningRealm
The name of the realm used for device provisioning.

environment
The platform environment of the app version: android, iphone, ...

link
The URL to access detail information about the application version.

publishUrl
For web applications, this is the URL under which the web application was published.

securityTest
The name of the security test for a protected resource.

supportRemoteDisable
true if the application version supports remote disabling.

supportsAuthenticity
true if the application version supports authenticatuib.

userAuthenticationRealm
The name of the realm used to authenticate users.

version
The version number of the app version.

versionLocked
Whether the version is locked.

The *applicationEnvironmentDataAccess* has the following properties:

action
The action to be done when a disabled app version is accessed. Possible values are: NOTIFY, BLOCK, NA.

createdTime
The date in ISO 8601 format when the app version access rule was created.

downloadLink
The download link where to obtain a new version of the application.

message
The message to be displayed when a disabled app version is accessed.

multiLanguageMessage
Internationalized variants of the message to be displayed when a disabled app version is accessed.

The *project* has the following properties:

name
    The name of the project, which is the context root of the runtime.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found.
```

500

```
An internal error occurred.
```

Device Application Status (PUT)

Changes the status of a specific application on a specific device.

Description

A device can be marked as enabled or disabled for a specific device. Disabled applications cannot access the server.

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*/applications/
application-name

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789/ap
```

Path Parameters

runtime-name
    The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

application-name
    The name of the application.

device-id
The device id.

Query Parameters

Query parameters are optional.

async
Whether the transaction is processed synchronously or asynchronously.
Allowed values are `true` and `false`. The default is synchronous processing.

locale
The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

Example as JSON

```
{
  "status" : "ENABLED",
}
```

Payload Properties

The payload has the following properties:

status
The status of the application: `ENABLED` or `DISABLED`.

Response

The meta data of the transaction.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "appName" : "myapplication",
      "deviceId" : "12345-6789",
      "status" : "ENABLED",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
```

```
      "status" : "FAILURE",
      "timeCreated" : "2014-04-13T00:18:36.979Z",
      "timeUpdated" : "2014-04-14T00:18:36.979Z",
      "type" : "CHANGE_DEVICE_APPLICATION_STATUS",
      "userName" : "demouser",
  },
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<set-applicationdevice-status-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="CHANGE_DEVICE_APPLICATION_STATUS"
    userName="demouser">
    <description
      appName="myapplication"
      deviceId="12345-6789"
      status="ENABLED"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-applicationdevice-status-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the status change.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
The date in ISO 8601 format when the adapter was created.

timeUpdated
The date in ISO 8601 format when the adapter was updated.

type
The type of the transaction, here always `CHANGE_DEVICE_APPLICATION_STATUS`.

userName
The user that initiated the transaction.

The *description* has the following properties:

appName
The application name.

deviceId
The device id.

status
The status of the application: `ENABLED` or `DISABLED`.

The *error* has the following properties:

details
The main error message.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

`The payload is invalid.`

403

`The user is not authorized to call this service.`

404

`The corresponding runtime or the device is not found.`

500

`An internal error occurred.`

Device (DELETE)

Deletes all meta information of a specific device.

Description

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightoperator**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789?asy

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

device-id
> The device id.

Query Parameters

Query parameters are optional.

async
> Whether the transaction is processed synchronously or asynchronously. Allowed values are true and false. The default is synchronous processing.

locale
> The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deleted device.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "deviceId" : "12345-6789",
      "status" : "LOST",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
```

```
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "REMOVE_DEVICE",
    "userName" : "demouser",
  },
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<remove-device-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="REMOVE_DEVICE"
    userName="demouser">
    <description
      deviceId="12345-6789"
      status="LOST"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</remove-device-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
> The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the device.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction, here always REMOVE_DEVICE.

userName
> The user that initiated the transaction.

The *description* has the following properties:

deviceId
> The device id.

status
> The status of the device: ACTIVE, LOST, STOLEN, EXPIRED, DISABLED.

The *error* has the following properties:

details
> The main error message.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime or the device is not found.
```

500

```
An internal error occurred.
```

Device Status (PUT)

Changes the status of a specific device.

Description

A device can be marked as active, lost, stolen, disabled, or expired. Lost, stolen or disabled devices cannot access the server. A device is marked expired if it has not connected to the Worklight server for 90 days.

This transaction can run synchronously or asynchronously. If processed asynchronously, the REST service returns before the transaction is completed. In this case, you can query the transaction result later with the transaction REST service.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789?

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web
> application, without the leading slash.

device-id
> The device id.

Query Parameters

Query parameters are optional.

async
> Whether the transaction is processed synchronously or asynchronously.
> Allowed values are true and false. The default is synchronous processing.

locale
> The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

Example as JSON

```
{
  "status" : "LOST",
}
```

Payload Properties

The payload has the following properties:

status
> The new status of the device: ACTIVE, LOST, STOLEN, EXPIRED, DISABLED.

Response

The meta data of the transaction.

Example as JSON

```
{
  "ok" : false,
  "productVersion" : "6.2.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "description" : {
      "deviceId" : "12345-6789",
      "status" : "LOST",
    },
    "errors" : [
      {
        "details" : "An internal error occured.",
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "FAILURE",
    "timeCreated" : "2014-04-13T00:18:36.979Z",
    "timeUpdated" : "2014-04-14T00:18:36.979Z",
    "type" : "CHANGE_DEVICE_STATUS",
    "userName" : "demouser",
  },
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<set-device-status-result
  ok="false"
  productVersion="6.2.0">
  <transaction
    appServerId="Tomcat"
    id="1"
    status="FAILURE"
    timeCreated="2014-04-13T00:18:36.979Z"
    timeUpdated="2014-04-14T00:18:36.979Z"
    type="CHANGE_DEVICE_STATUS"
    userName="demouser">
    <description
      deviceId="12345-6789"
      status="LOST"/>
    <errors>
      <error details="An internal error occured."/>
      ...
    </errors>
    <project name="myproject"/>
  </transaction>
</set-device-status-result>
```

Response Properties

The response has the following properties:

ok Whether the transaction was successful.

productVersion
    The exact product version.

transaction
> The details of the transaction.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the status change.

errors
> The errors occurred during the transaction.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction, here always CHANGE_DEVICE_STATUS.

userName
> The user that initiated the transaction.

The *description* has the following properties:

deviceId
> The device id.

status
> The status of the device: ACTIVE, LOST, STOLEN, EXPIRED, DISABLED.

The *error* has the following properties:

details
> The main error message.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

400

The payload is invalid.

403

The user is not authorized to call this service.

404

The corresponding runtime or the device is not found.

500

An internal error occurred.

Devices (GET)

Retrieves meta information for the list of devices that accessed this project.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/devices

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices?locale=de_DE&c

Path Parameters

runtime-name
   The name of the runtime. This is the context root of the runtime web
   application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
   The locale used for error messages.

offset
   The offset from the beginning of the list if only a part of the list (a page)
   should be returned.

orderBy
   The sort mode. By default, the elements are sorted in increasing order. If the
   sort mode starts with - (minus sign), the elements are sorted in decreasing
   order. Possible sort modes are: uid, friendlyName, deviceModel,
   deviceEnvironment, status, lastAccessed. The default sort mode is: uid.

pageSize
   The number of elements if only a part of the list (a page) should be returned.
   The default value is 100.

query
   A device friendly name or a user to search for.

Produces

application/json, application/xml, text/xml

Response

The meta data of the devices that accessed this project.

Example as JSON

```
{
  "items" : [
    {
      "applicationDeviceAssociations" : [
        {
          "appName" : "myapplication",
          "deviceId" : "12345-6789",
          "deviceStatus" : "LOST",
          "status" : "ENABLED",
        },
        ...
      ],
      "deviceEnvironment" : "iphone",
      "deviceModel" : "Nexus 7",
      "deviceOs" : "4.4",
      "friendlyName" : "Jeremy's Personal Phone",
      "id" : "12345-6789",
      "lastAccessed" : "2014-05-13T00:18:36.979Z",
      "status" : "LOST",
      "uid" : "Jeremy",
    },
    ...
  ],
  "pageSize" : 100,
  "productVersion" : "6.2.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<devices
  pageSize="100"
  productVersion="6.2.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deviceEnvironment="iphone"
      deviceModel="Nexus 7"
      deviceOs="4.4"
      friendlyName="Jeremy's Personal Phone"
      id="12345-6789"
      lastAccessed="2014-05-13T00:18:36.979Z"
      status="LOST"
      uid="Jeremy">
      <applicationDeviceAssociations>
        <applicationDeviceAssociation
          appName="myapplication"
          deviceId="12345-6789"
          deviceStatus="LOST"
          status="ENABLED"/>
        ...
      </applicationDeviceAssociations>
```

```
        </item>
        ...
    </items>
</devices>
```

Response Properties

The response has the following properties:

items
: The array of device meta information

pageSize
: The page size if only a page of devices is returned.

productVersion
: The exact product version.

startIndex
: The start index in the total list if only a page of devices is returned.

totalListSize
: The total number of devices.

The *device* has the following properties:

applicationDeviceAssociations
: The applications on the device.

deviceEnvironment
: The platform environment of the app version: iphone, android, ...

deviceModel
: The device model.

deviceOs
: The device operating system.

friendlyName
: The friendly name of the device.

id The device id.

lastAccessed
: The date in ISO 8601 format when the device was last accessed.

status
: The status of the device: ACTIVE, LOST, STOLEN, EXPIRED, DISABLED.

uid
: The user name of the device.

The *device application* has the following properties:

appName
: The name of the application.

deviceId
: The device id.

deviceStatus
: The status of the device:ACTIVE, LOST, STOLEN, EXPIRED, DISABLED.

status
: The status of the application: ENABLED or DISABLED.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found or not running.
```

500

```
An internal error occurred.
```

Event Source (GET)

Retrieves meta information for the event source.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/eventsources/*adapter-name*/*eventsource-name*

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/event
```

Path Parameters

runtime-name
　The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

adapter-name
　The name of the adapter.

eventsource-name
　The name of the event source.

Query Parameters

Query parameters are optional.

locale
　The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the event source.

Example as JSON

```
{
  "numberOfMessagesSent" : 1,
  "numberOfSubscribedUsers" : 1,
  "productVersion" : "6.2.0",
  "qname" : "SampleAdapter.SampleEventSource",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<eventsources
  numberOfMessagesSent="1"
  numberOfSubscribedUsers="1"
  productVersion="6.2.0"
  qname="SampleAdapter.SampleEventSource"/>
```

Response Properties

The response has the following properties:

numberOfMessagesSent
> Number of messages sent to this event source.

numberOfSubscribedUsers
> Number of subscribed users of this event source.

productVersion
> The exact product version.

qname
> The name of the event source.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found.
```

500

```
An internal error occurred.
```

Event Sources (GET)

Retrieves meta information for the list of event sources.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/eventsources

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/event

Path Parameters

`runtime-name`
  The name of the runtime. This is the contex root of the runtime web
  application, without the leading slash.

Query Parameters

Query parameters are optional.

`locale`
  The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the event sources.

Example as JSON

```
{
  "eventsources" : [
    {
      "numberOfMessagesSent" : 1,
      "numberOfSubscribedUsers" : 1,
      "qname" : "myadapter.myeventsource",
    },
    ...
  ],
  "productVersion" : "6.2.0",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<eventsources productVersion="6.2.0">
  <eventsources>
    <eventsource
      numberOfMessagesSent="1"
      numberOfSubscribedUsers="1"
      qname="myadapter.myeventsource"/>
    ...
  </eventsources>
</eventsources>
```

Response Properties

The response has the following properties:

eventsources
　　The array of event source meta information

productVersion
　　The exact product version.

The *eventsource* has the following properties:

numberOfMessagesSent
　　Number of messages sent to this event source.

numberOfSubscribedUsers
　　Number of subscribed users of this event source.

qname
　　The name of the event source.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found.
```

500

```
An internal error occurred.
```

Mediator (GET)

Retrieves meta information of the mediator.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/mediators/*mediator-name*

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/mediato
```

Path Parameters

runtime-name
　　The name of the runtime. This is the contex root of the runtime web
　　application, without the leading slash.

mediator-name
   The name of the mediator.

Query Parameters

Query parameters are optional.

locale
   The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the mediator.

Example as JSON

```
{
  "productVersion" : "6.2.0",
  "type" : "Google",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<mediators
  productVersion="6.2.0"
  type="Google"/>
```

Response Properties

The response has the following properties:

productVersion
   The exact product version.

type
   The type of the mediator.

Errors

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime is not found.
```

500
```
An internal error occurred.
```

Mediators (GET)

Retrieves the list of all supported mediators for sending notifications.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/mediators

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/mediators

Path Parameters

runtime-name
    The name of the runtime. This is the contex root of the runtime web
    application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The list of all supported mediators for sending notifications.

Example as JSON

```
{
  "mediators" : [
    {
      "type" : "Google",
    },
    ...
  ],
  "productVersion" : "6.2.0",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<mediators productVersion="6.2.0">
  <mediators>
```

```
      <mediator type="Google"/>
      ...
    </mediators>
</mediators>
```

Response Properties

The response has the following properties:

mediators
    The array of mediator meta information

productVersion
    The exact product version.

The *mediator* has the following properties:

type
    The type of the mediator.

Errors

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime is not found.
```

500
```
An internal error occurred.
```

Push Enabled Applications (GET)

Retrieves meta information for the list of deployed push enabled applications.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/appli
```

Path Parameters

`runtime-name`
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

`locale`
> The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the deployed push enabled applications.

Example as JSON

```
{
  "applications" : [
    {
      "applicationEnvironments" : [
        {
          "deviceCount" : 1,
          "mediatorType" : "Google",
          "numberOfMessagesSent" : 1,
          "userCount" : 1,
        },
        ...
      ],
      "deviceCount" : 1,
      "displayName" : "SampleApplication",
      "userCount" : 1,
    },
    ...
  ],
  "productVersion" : "6.2.0",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<pushnotification productVersion="6.2.0">
  <applications>
    <application
      deviceCount="1"
      displayName="SampleApplication"
      userCount="1">
      <applicationEnvironments>
        <applicationEnvironment
          deviceCount="1"
          mediatorType="Google"
          numberOfMessagesSent="1"
          userCount="1"/>
        ...
      </applicationEnvironments>
    </application>
    ...
  </applications>
</pushnotification>
```

Response Properties

The response has the following properties:

applications
    The array of push enabled application meta information

productVersion
    The exact product version.

The *application* has the following properties:

applicationEnvironments
    The array of application environments.

deviceCount
    Number of subscribed devices of this application.

displayName
    The name of the application.

userCount
    Number of subscribed users of this application.

The *applicationEnvironment* has the following properties:

deviceCount
    Number of subscribed devices of this application environment.

mediatorType
    The name of the application environment.

numberOfMessagesSent
    Number of messages sent for this application environment.

userCount
    Number of subscribed users of this application environment.

Errors

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime is not found.
```

500
```
An internal error occurred.
```

Runtime (DELETE)

Deletes a specific runtime.

Description

The purpose of this API is to allow to cleanup the data base. It is only possible to delete a runtime when it is stopped. A runtime that is currently active cannot be deleted.

Roles

Users in the following roles are authorized to perform this operation:
* **worklightadmin**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime?locale=de_DE

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web
> application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
> The locale used for error messages.

Produces

application/json, application/xml, text/xml

Errors

403
The user is not authorized to call this service.

409
The corresponding runtime cannot be deleted since it is still running. You must stop the runtime firs

500
An internal error occurred.

Runtime (GET)

Retrieves meta information for a specific runtime.

Roles

Users in the following roles are authorized to perform this operation:
* **worklightadmin**
* **worklightdeployer**
* **worklightmonitor**
* **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime?expand=true&locale=

Path Parameters

`runtime-name`
    The name of the runtime. This is the contex root of the runtime web
    application, without the leading slash.

Query Parameters

Query parameters are optional.

`expand`
    Set to `true` to show details of the applications and adapters. The default is
    `false`

`locale`
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta information for the runtime.

Example as JSON

```
{
  "adapters" : [
    {
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "description" : "My first sample adapter",
      "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/ada
      "name" : "myadapter",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "procedures" : [ "getSomething", ... ],
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
    },
    ...
  ],
  "applications" : [
    {
      "description" : "My first sample application",
      "displayName" : "My Sample Application",
      "environments" : [
        {
```

```
            "applicationEnvironmentDataAccess" : {
              "action" : "NOTIFY",
              "createdTime" : "2014-04-13T00:18:36.979Z",
              "message" : "This version is no longer supported.",
            },
            "authenticityCheckRule" : "DISABLED",
            "buildTime" : "2014-03-29T00:18:36.979Z",
            "deployTime" : "2014-04-13T00:18:36.979Z",
            "deviceProvisioningRealm" : "myProvRealm",
            "environment" : "android",
            "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/ap
            "securityTest" : "mobileTest",
            "supportRemoteDisable" : true,
            "supportsAuthenticity" : true,
            "userAuthenticationRealm" : "myAuthRealm",
            "version" : "1.0",
            "versionLocked" : false,
          },
          ...
        ],
        "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applic
        "name" : "myapplication",
        "platformVersion" : "6.1.0.00.20131126-0630",
        "projects" : [
          {
            "name" : "myproject",
          },
          ...
        ],
      },
      ...
    ],
    "auditEnabled" : true,
    "bitlyApikey" : "",
    "bitlyUsername" : "",
    "name" : "myruntime",
    "numberOfActiveDevices" : 100,
    "numberOfDecommisionedDevices" : 5,
    "platformVersion" : "6.1.0.00.20131126-0630",
    "productVersion" : "6.2.0",
    "running" : true,
    "serverVersion" : "6.2.0",
}
```

Example as XML

```xml
<?xml version="1.0" encoding="UTF-8"?>
<runtime
  auditEnabled="true"
  bitlyApikey=""
  bitlyUsername=""
  name="myruntime"
  numberOfActiveDevices="100"
  numberOfDecommisionedDevices="5"
  platformVersion="6.1.0.00.20131126-0630"
  productVersion="6.2.0"
  running="true"
  serverVersion="6.2.0">
  <adapters>
    <adapter
      deployTime="2014-04-13T00:18:36.979Z"
      description="My first sample adapter"
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/r
      name="myadapter"
      platformVersion="6.1.0.00.20131126-0630">
      <procedures>
        <procedure>getSomething</procedure>
```

```
          ...
        </procedures>
        <projects>
          <project name="myproject"/>
          ...
        </projects>
      </adapter>
    ...
  </adapters>
  <applications>
    <application
      description="My first sample application"
      displayName="My Sample Application"
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
      name="myapplication"
      platformVersion="6.1.0.00.20131126-0630">
      <environments>
        <environment
          authenticityCheckRule="DISABLED"
          buildTime="2014-03-29T00:18:36.979Z"
          deployTime="2014-04-13T00:18:36.979Z"
          deviceProvisioningRealm="myProvRealm"
          environment="android"
          link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/app
          securityTest="mobileTest"
          supportRemoteDisable="true"
          supportsAuthenticity="true"
          userAuthenticationRealm="myAuthRealm"
          version="1.0"
          versionLocked="false">
          <applicationEnvironmentDataAccess
            action="NOTIFY"
            createdTime="2014-04-13T00:18:36.979Z"
            message="This version is no longer supported."/>
        </environment>
        ...
      </environments>
      <projects>
        <project name="myproject"/>
        ...
      </projects>
    </application>
    ...
  </applications>
</runtime>
```

Response Properties

The response has the following properties:

adapters
    The array of adapters (shown only with expand=true).

applications
    The array of applications (shown only with expand=true).

auditEnabled
    Whether audit is enabled.

bitlyApikey
    The key for the Bitly service.

bitlyUsername
    The user name for the Bitly service.

name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

numberOfActiveDevices
> The number of active devices using this runtime.

numberOfAdapters
> The number of adapters deployed in this runtime (shown only with `expand=false`).

numberOfApplications
> The number of applications deployed in this runtime (shown only with `expand=false`).

numberOfDecommisionedDevices
> The number of devices decommissioned for this runtime.

platformVersion
> The exact version number of the Worklight development tools (Studio) that built the project WAR file.

productVersion
> The exact product version.

running
> Whether the runtime is currently active or has stopped.

serverVersion
> The exact Worklight Server version number from which `worklight-jee-library.jar` is taken.

The *adapter* has the following properties:

deployTime
> The date in ISO 8601 format when the adapter was deployed.

description
> The description of the adapter.

link
> The URL to access detail information about the adapter.

name
> The name of the adapter.

platformVersion
> The exact version number of the Worklight development tools (Studio) that built the adapter.

procedures
> The JavaScript procedures of the adapter.

projects
> The projects the adapter belong to.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

The *application* has the following properties:

description
>   The description of the application.

displayName
>   The display name of the application.

environments
>   The array of application environments.

link
>   The URL to access detail information about the application.

name
>   The name of the application.

platformVersion
>   The exact version number of the Worklight development tools (Studio) that built the application.

projects
>   The projects the application belong to.

The *environment* has the following properties:

applicationEnvironmentDataAccess
>   The access rule to be executed when the app version is disabled.

authenticityCheckRule
>   Whether the authenticity is checked. Possible values are: ENABLED, IGNORED, DISABLED.

buildTime
>   The time stamp when the app version was built.

deployTime
>   The date in ISO 8601 format when the adapter was deployed.

deviceProvisioningRealm
>   The name of the realm used for device provisioning.

environment
>   The platform environment of the app version: android, iphone, ...

link
>   The URL to access detail information about the application version.

publishUrl
>   For web applications, this is the URL under which the web application was published.

securityTest
>   The name of the security test for a protected resource.

supportRemoteDisable
>   true if the application version supports remote disabling.

supportsAuthenticity
>   true if the application version supports authenticatuib.

userAuthenticationRealm
>   The name of the realm used to authenticate users.

version
>   The version number of the app version.

versionLocked
    Whether the version is locked.

The *applicationEnvironmentDataAccess* has the following properties:

action
    The action to be done when a disabled app version is accessed. Possible values
    are: NOTIFY, BLOCK, NA.

createdTime
    The date in ISO 8601 format when the app version access rule was created.

downloadLink
    The download link where to obtain a new version of the application.

message
    The message to be displayed when a disabled app version is accessed.

multiLanguageMessage
    Internationalized variants of the message to be displayed when a disabled app
    version is accessed.

Errors

403

The user is not authorized to call this service.

404

The corresponding runtime is not found.

500

An internal error occurred.

Runtime Lock (DELETE)

Forces the release of the transaction lock of a runtime.

Description

This API should not be used in normal operations.

Transactions are performed sequentually. Hence each transaction such as deploying
an application or adapter takes the runtime lock. The next transaction waits until
the lock is released. After a serious crash, it may happen that the lock is still taken
even though the corresponding transaction crashed. The lock will get automatically
released after 30 minutes. However, with this API, you can force the release of the
lock earlier.

Forcing the release of the lock when a transaction is currently active may corrupt
the system. You should use this API only when you are sure that no transaction is
currently active.

Roles

Users in the following roles are authorized to perform this operation:

• **worklightadmin**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/lock

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/lock?locale=de_DE

Path Parameters

runtime-name
    The name of the runtime. This is the contex root of the runtime web
    application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

Example as JSON

```
{
  "busy" : false,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<lock busy="false"/>
```

Response Properties

The response has the following properties:

busy
    Whether the runtime is still busy with a transaction after forcing the release of
    the lock.

Errors

403
The user is not authorized to call this service.

500
An internal error occurred.

Runtime Lock (GET)

Retrieves information about the transaction lock of a runtime.

Description

Transactions are performed sequentually. Hence each transaction such as deploying an application or adapter takes the runtime lock. The next transaction waits until the lock is released. This API allowed to retrieve whether a runtime is currently busy with a transaction.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/lock

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/lock?locale=de_DE
```

Path Parameters

runtime-name
    The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

Example as JSON

```
{
  "busy" : true,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<lock busy="true"/>
```

Response Properties

The response has the following properties:

busy
    Whether the runtime is currently busy with a transaction.

Errors

403
The user is not authorized to call this service.

500
An internal error occurred.

Runtimes (GET)

Retrieves meta information for the list of runtimes.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes?locale=de_DE&mode=db

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

mode
    The default mode running retrieves only the running runtimes, while the mode db retrieves also the runtimes stored in the data base that might not be running.

Produces

application/json, application/xml, text/xml

Response

The meta information for the list of runtimes.

Example as JSON

```
{
  "productVersion" : "6.2.0",
  "projects" : [
    {
      "link" : "https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime",
      "name" : "myruntime",
      "numberOfActiveDevices" : 100,
      "numberOfAdapters" : 1,
      "numberOfApplications" : 1,
      "numberOfDecommisionedDevices" : 5,
      "running" : true,
    },
    ...
  ],
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<projectconfiguration productVersion="6.2.0">
  <projects>
    <project
      link="https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime"
      name="myruntime"
      numberOfActiveDevices="100"
      numberOfAdapters="1"
      numberOfApplications="1"
      numberOfDecommisionedDevices="5"
      running="true"/>
    ...
  </projects>
</projectconfiguration>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

projects
The array of runtimes.

The *runtime* has the following properties:

link
The URL to access detail information about the runtime.

name
The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

numberOfActiveDevices
The number of active devices using this runtime.

numberOfAdapters
The number of adapters deployed in this runtime.

numberOfApplications
The number of applications deployed in this runtime.

numberOfDecommisionedDevices
   The number of devices decommissioned for this runtime.

running
   Whether the runtime is currently active or has stopped.

Errors

403

```
The user is not authorized to call this service.
```

500

```
An internal error occurred.
```

Subscriptions (GET)

Retrieves meta information of the subscriptions.

Description

The subscriptions can be obtained for application, for a particular tag, for a particular devieId and a combination of application, tag name and deviceId

Roles

Users in the following roles are authorized to perform this operation:
* **worklightadmin**
* **worklightdeployer**
* **worklightmonitor**
* **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/
application-name/subscriptions

Example

```
https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/appli
```

Path Parameters

runtime-name
   The name of the runtime. This is the contex root of the runtime web
   application, without the leading slash.

application-name
   The name of the application.

Query Parameters

Query parameters are optional.

locale
    The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The meta data of the subscriptions.

Example as JSON

```
{
  "productVersion" : "6.2.0",
  "subscriptions" : [
    {
      "deviceId" : "testdevice",
      "tagName" : "testtag",
    },
    ...
  ],
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<subscriptions productVersion="6.2.0">
  <subscriptions>
    <subscription
      deviceId="testdevice"
      tagName="testtag"/>
    ...
  </subscriptions>
</subscriptions>
```

Response Properties

The response has the following properties:

productVersion
    The exact product version.

subscriptions
    The array of subscription meta information

The *subscription* has the following properties:

deviceId
    The unique id of the device.

tagName
    The name of the tag.

Errors

403
```
The user is not authorized to call this service.
```

404
```
The corresponding runtime is not found.
```

500

`An internal error occurred.`

Transaction (GET)

Retrieves information of a specific transaction.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

`GET`

Path

/management-apis/1.0/runtimes/*runtime-name*/transactions/*transaction-id*

Example

`https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/transactions/1?loca`

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

transaction-id
> The transaction id.

Query Parameters

Query parameters are optional.

locale
> The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The information of the specified transaction.

Example as JSON

```
{
  "appServerId" : "Tomcat",
  "description" : {
  },
  "errors" : [
    {
```

```
      "details" : "An internal error occured.",
    },
    ...
  ],
  "id" : 1,
  "productVersion" : "6.2.0",
  "project" : {
    "name" : "myproject",
  },
  "status" : "FAILURE",
  "timeCreated" : "2014-04-13T00:18:36.979Z",
  "timeUpdated" : "2014-04-14T00:18:36.979Z",
  "type" : "DELETE_ADAPTER",
  "userName" : "demouser",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction
  appServerId="Tomcat"
  id="1"
  productVersion="6.2.0"
  status="FAILURE"
  timeCreated="2014-04-13T00:18:36.979Z"
  timeUpdated="2014-04-14T00:18:36.979Z"
  type="DELETE_ADAPTER"
  userName="demouser">
  <description/>
  <errors>
    <error details="An internal error occured."/>

    ...
  </errors>
  <project name="myproject"/>
</transaction>
```

Response Properties

The response has the following properties:

appServerId
> The id of the web application server.

description
> The details of the transaction, depending on the transaction type.

errors
> The errors occured during the transacton.

id The id of the transaction.

productVersion
> The exact product version.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
The date in ISO 8601 format when the adapter was updated.

type
The type of the transaction.

userName
The user that initiated the transaction.

The *error* has the following properties:

details
The main error message.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

403
The user is not authorized to call this service.

404
The corresponding runtime or the transaction is not found.

500
An internal error occurred.

Transactions (GET)

Retrieves information of all transaction.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/runtimes/*runtime-name*/transactions

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/transactions?file=f

Path Parameters

runtime-name
> The name of the runtime. This is the contex root of the runtime web application, without the leading slash.

Query Parameters

Query parameters are optional.

file
> If `true`, the transactions are delivered as zip file. In this case, paging parameters are ignored.

locale
> The locale used for error messages.

offset
> The offset from the beginning of the list if only a part of the list (a page) should be returned.

orderBy
> The sort mode. By default, the elements are sorted in increasing order. If the sort mode starts with - (minus sign), the elements are sorted in decreasing order. Possible sort modes are: created, updated, type, status, user, server. The default sort mode is: created.

pageSize
> The number of elements if only a part of the list (a page) should be returned. The default value is 100.

Produces

application/json, application/xml, text/xml, application/zip

Response

The information of the transactions.

Example as JSON

```
{
  "items" : [
    {
      "appServerId" : "Tomcat",
      "description" : {
      },
      "errors" : [
        {
          "details" : "An internal error occured.",
        },
        ...
      ],
      "id" : 1,
      "project" : {
        "name" : "myproject",
      },
      "status" : "FAILURE",
      "timeCreated" : "2014-04-13T00:18:36.979Z",
      "timeUpdated" : "2014-04-14T00:18:36.979Z",
      "type" : "DELETE_ADAPTER",
      "userName" : "demouser",
    },
```

```
    ...
  ],
  "pageSize" : 100,
  "productVersion" : "6.2.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions
  pageSize="100"
  productVersion="6.2.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      appServerId="Tomcat"
      id="1"
      status="FAILURE"
      timeCreated="2014-04-13T00:18:36.979Z"
      timeUpdated="2014-04-14T00:18:36.979Z"
      type="DELETE_ADAPTER"
      userName="demouser">
      <description/>
      <errors>
        <error details="An internal error occured."/>
        ...
      </errors>
      <project name="myproject"/>
    </item>
    ...
  </items>
</transactions>
```

Response Properties

The response has the following properties:

items
> The array of transations

pageSize
> The page size if only a page of transactions is returned.

productVersion
> The exact product version.

startIndex
> The start index in the total list if only a page of transactions is returned.

totalListSize
> The total number of transactions.

The *transaction* has the following properties:

appServerId
> The id of the web application server.

description
> The details of the transaction, depending on the transaction type.

errors
> The errors occured during the transacton.

id The id of the transaction.

project
> The current project.

status
> The state of the transaction: PENDING, PREPARING, COMMITTING, REJECTING, SUCCESS, FAILURE, CANCELED. Synchronous transactions can have the state SUCCESS and FAILURE. Asynchronous transactions can also have the other states.

timeCreated
> The date in ISO 8601 format when the adapter was created.

timeUpdated
> The date in ISO 8601 format when the adapter was updated.

type
> The type of the transaction.

userName
> The user that initiated the transaction.

The *error* has the following properties:

details
> The main error message.

The *project* has the following properties:

name
> The name of the project, which is the context root of the runtime.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found.
```

500

```
An internal error occurred.
```

Unsubscribe SMS (POST)

Unsubscribes the list of given phone numbers for SMS.

Roles

Users in the following roles are authorized to perform this operation:
- **worklightadmin**
- **worklightdeployer**

Method

POST

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/unsubscribeSMS

Example

https://www.myserver.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/unsub

Path Parameters

runtime-name
 The name of the runtime. This is the contex root of the runtime web
 application, without the leading slash.

Query Parameters

Query parameters are optional.

locale
 The locale used for error messages.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

The payload with comma separated list of phone numbers.

Example as JSON

```
{
  "numbers" : "1234,5678",
}
```

Payload Properties

The payload has the following properties:

numbers
 Comma separated list of phone numbers.

Response

The response status of SMS unsubscription.

Example as JSON

```
{
  "failure" : "5678",
  "success" : "1234",
}
```

Example as XML

```
<?xml version="1.0" encoding="UTF-8"?>
<unsubscribeSMS
  failure="5678"
  success="1234"/>
```

Response Properties

The response has the following properties:

failure
> Comma separated list of phone numbers which are not deleted.

success
> Comma separated list of phone numbers which are successfully deleted.

Errors

403

```
The user is not authorized to call this service.
```

404

```
The corresponding runtime is not found.
```

500

```
An internal error occurred.
```

Chapter 12. Deploying Worklight projects

After you have created projects and apps with Worklight Studio, you must deploy them to the production environment.

Note: Before you can deploy the project to the production environment, you must install the Worklight Administration Components as described in "Installing and configuring Worklight Server administration" on page 104.

You can deploy several Worklight runtime environments (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.

Note: A Worklight Administration Service must be installed on the application server where you install a Worklight runtime environment. Otherwise, the Worklight runtime environment cannot download its applications and adapters and cannot start.

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 and the same Worklight Administration Service. However, they must be migrated to be compatible with the current version of the Worklight Administration service. For more information about migration, see "Migrating a project WAR file for use with a new Worklight Server" on page 1164.

For more information, see "Separation of lifecycle between Worklight Server and Worklight Studio" on page 345.

Read this series of topics to learn how to deploy your Worklight projects and apps to the production environment.

Deploying 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 that contains one or more applications in Worklight Studio. A WAR file and a set of `.wlapp` files are created in the `bin` folder of your 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 that is 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 that is called *app-name-env-version*`.wlapp` is created per environment. For example: `myApp-android-1.0.wlapp`.

About this task

First, you prepare the application or applications for deployment, and then you deploy them. You can deploy many apps within the same project. The following instructions lead you through this process.

Procedure

1. Install the Worklight Server administration components as described in "Installing and configuring Worklight Server administration" on page 104. You can have several Worklight runtime environments managed by the same Worklight Console. Verify that you have deployment rights for Worklight, such as the role of **worklightdeployer** or **worklightadmin**. For more information, see "Configuring user authentication for Worklight Server administration" on page 151.

2. 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 430

3. You might want to look at the settings in the `worklight.properties` file, which is in `server/conf`. Those settings define the default values for the 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 a Worklight project in production by using JNDI environment entries" on page 1182.

4. Build each application in either of two ways:
   - Right-click the application and click **Run As** > **Build All Environments**.
   - Use the Ant script tool that is described in "Ant tasks for building and deploying applications and adapters" on page 1190

   If you use Worklight Studio, the project WAR file is named `projectName`.war and is in the \bin folder. This file contains the project configuration that was done in steps 1 and 2 and any classes that are built from Java code in the server/java folder.

5. 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, or undeploying Worklight Server by using the Server Configuration Tool" on page 1106
   - 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 1114.
     - 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 1141.
     - The list of JNDI environment entries that can be configured is at "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.
     - Sample Ant files using these Ant tasks can be found in the 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 1156.

       You must call **configuredatabase** first (target databases in the sample Ant files) and then **configureapplicationserver** (target install in the sample Ant files).

6. Open the Worklight Console of the target environment. If the Worklight Console is installed with the default context root, its URL is of the form https://*your-remote-server*:*server-port*/worklightconsole, or if HTTPS is not supported in your application server, it is the insecure URL http://*your-remote-server*:*server-port*/worklightconsole.

   Important: If you access the Worklight Console through HTTP instead of HTTPS, your Worklight administration user password is compromised.

7. From the Worklight Console, deploy the relevant .wlapp files from the bin folder of your Worklight project.
   - For more information about how to deploy an application by using Worklight Console, see "Deploying apps" on page 1199.
   - You can also deploy the app to the target environment by using the Worklight Server administration command-line tools. For more information about how to deploy an app by using the provided command-line tools, see "Administering Worklight applications through ANT" on page 1258 and "Administering Worklight applications through the command line" on page 1275.

8. Deploy the adapters from the development environment.

a. Navigate to the `bin` folder in your project.
b. Copy the `.adapter` file or files.
c. From the Worklight Console, deploy the `.adapter` files from the `bin` folder of your project.
   - For more information about how to deploy an adapter by using Worklight Console, see "Deploying adapters" on page 1200.
   - You can also deploy the adapter to the target environment by using the Worklight Server administration command-line tools. For more information about how to deploy an app by using the provided command-line tools, see "Administering Worklight applications through ANT" on page 1258 and "Administering Worklight applications through the command line" on page 1275.

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 Foundation" on page 13.

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 a Worklight project WAR file.

Note: Since IBM Worklight Foundation V6.2.0, the `worklight-ant-builder.jar` file is included in the Command Line Interface for IBM Worklight Developers, whereas in earlier versions, it was included in Worklight Server. By default, `worklight-ant-builder.jar` is installed in the following location: *<CLI Install Path>*/public/worklight-ant-builder.jar. For example, on OSX, the default CLI Install Path is /Applications/IBM/Worklight-CLI. If you use the default installation path, the Ant task is installed here: /Applications/IBM/Worklight-CLI/public/worklight-ant-builder.jar.

Building the project WAR file

The Ant task for building the project WAR file has the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project name="myProject" default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_CLI_INSTALL_DIR/public/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

For the Worklight runtime component to start, you must deploy it to the server where the Worklight administration service is installed. If you use WebSphere Application Server Network Deployment, you can alternatively install the runtime component in a server or cluster of the cell other than the one where the administration service that manages this runtime is installed. If you do so, you must start the server or cluster where the administration service is installed before the one where the runtime component is installed.

When you work with a farm of application servers, declare the server farm as described in "Installing a server farm" on page 164. The database and application server prerequisites for this task are described in "Installation prerequisites" on page 71.

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" on page 1104. The WAR file contains the default configuration values for the server, and some resources for the Worklight applications and adapters.

- For project WARs built with versions earlier than V6.2.0.x: The project WAR file must be built with the same version of Worklight Studio as the version used to build the apps that are deployed on the Worklight Server.
- For project WAR files that were built with versions 6.2.0 and later, and deployed to Worklight Server V6.2.0.1, apps and adapters that were built with any version, 5.0.6.x and above (but not later than the project WAR version itself), can be deployed.

You can deploy a Worklight project in one the following ways:

- By using the Server Configuration Tool
- By using a set of Ant tasks that are supplied with Worklight Server to deploy a project WAR file and configure your databases and application servers.
- By following the procedure for "Creating and configuring the databases manually" on page 1121 and for deploying the Worklight Console manually.

Deploying, updating, or undeploying Worklight Server by using the Server Configuration Tool

The Server Configuration Tool is a graphical tool that you can use to deploy, update, or undeploy a Worklight Server to or from an Application Server and database.

If you use this tool in production to upgrade a Worklight Server, you must complete more actions to upgrade the server, as described in "Upgrading to Worklight Server V6.2.0 in a production environment" on page 355.

Before you use this tool, verify that the user who runs the Server Configuration Tool has the privileges that are described in "File system prerequisites" on page 72.

The Server Configuration Tool provides the same capabilities as the Ant tasks that are described in "Ant tasks for deploying a project WAR file and configuring an application server" on page 1109 but the tool is limited to a set of operations that are described in the following paragraphs, which 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 that are listed in "Configuring a Worklight project in production by using JNDI environment entries" on page 1182. To define those properties, use Ant files. 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 1109.)
- The Server Configuration Tool must be started on the computer where your application server is installed.
- The Server Configuration Tool maintains a deployment status of configuration server components, whether they are deployed or not. This status is not accurate if the Worklight Server components are 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 in this environment.
- You can use the Server Configuration Tool to install Worklight Server to WebSphere Application Server Network Deployment (clusters, servers), to a stand-alone WebSphere Application Server instance, or to a Liberty or Tomcat server. However, you cannot use the Server Configuration Tool to install Worklight Server to a server farm.
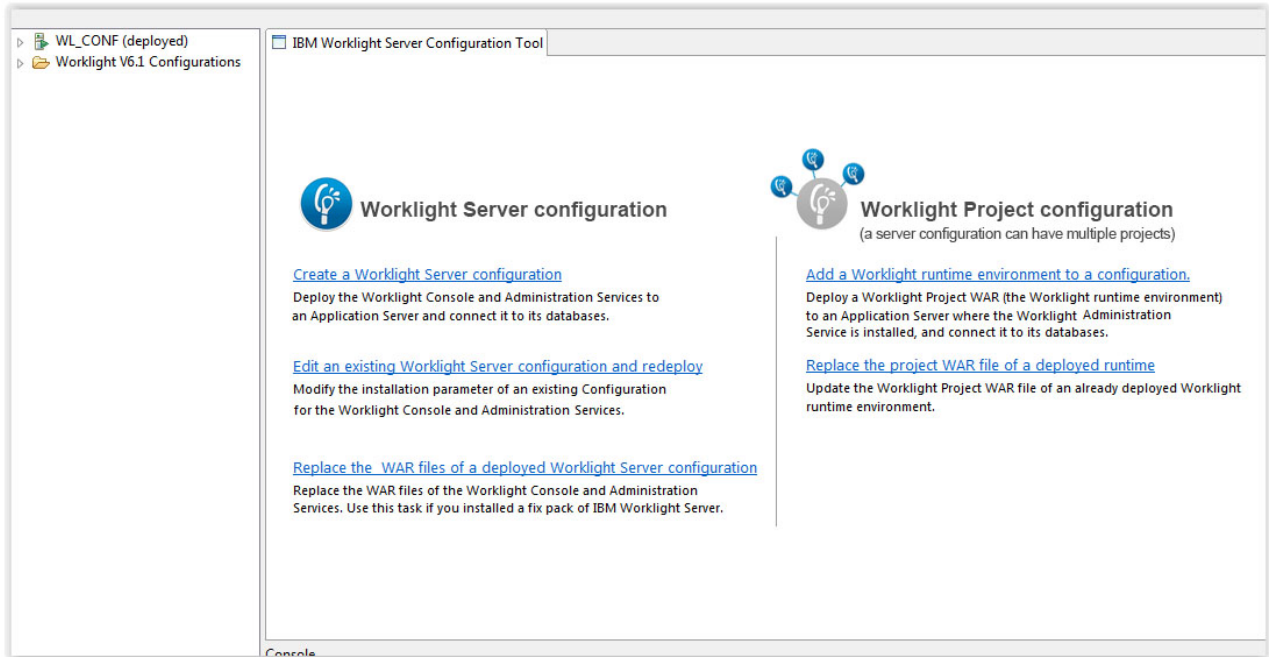
Figure 132. Server Configuration Tool main window

Running the Server Configuration Tool

You can start the 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, run the command `WL_INSTALL_DIR`/shortcuts/configuration-tool.sh.
- On Windows:
  - By using the **Start** > **Server Configuration Tool** menu command.
  - In Windows Explorer, double-click the file `WL_INSTALL_DIR`/shortcuts/configuration-tool.bat.
  - In a console window, run `WL_INSTALL_DIR`/shortcuts/configuration-tool.bat.
- On Mac OS X

  Note: Worklight Server is not supported for production in this environment.
  - In the Finder, double-click the file `WL_INSTALL_DIR`/shortcuts/configuration-tool.sh.
  - In a Terminal window, run `WL_INSTALL_DIR`/shortcuts/configuration-tool.sh.

Main tasks

Create a Worklight Server configuration

For more information, see "Installing Worklight Server administration with the Server Configuration Tool" on page 111.

After a Worklight Server configuration is created, you can do the following tasks:

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, the work flow is as follows:

1. You are prompted to select one of the configurations visible in the Navigation view.
2. If passwords are required to redeploy the configuration, you are prompted to enter them.
3. After you enter the passwords, the configuration is checked for errors.
4. If errors are found, a report is displayed.
5. You can then edit the configuration.
6. If the configuration contains no errors, the **Redeploy** button is enabled.
7. When you click **Redeploy**, the Worklight administration components are uninstalled from the application server, and reinstalled with the new parameters.

Add a Worklight runtime environment to a configuration
Use this task to add a Worklight runtime environment to a Worklight Server configuration.

To create a new Worklight runtime environment, complete the following steps:

1. Select a Worklight Server configuration.
2. Select **File** > **Add Worklight runtime environment**.
3. Enter a descriptive name for the Worklight runtime environment.
4. Select the path for the Worklight project WAR file to be deployed.
5. Step through the wizard to describe the target database management system.

   If you need to create a database for your Worklight runtime environment, the Server Configuration Tool can create it for you. If you provide the requested administrator password when you are prompted in the Database Creation Request panel, the database for Worklight runtime environment is created. 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 1110.
6. After you provide all the necessary information, the **Deploy** button is enabled. When you click **Deploy**, the following effects take place:

   a. The configuration file is saved.
   b. If the database contains no Worklight tables, the Worklight tables are created.
   c. If the database contains Worklight tables for an older version of Worklight, the tables are upgraded to the current Worklight version.
   d. If the database operations succeed, the Worklight Server is deployed to the application server.
   e. If the WAR file needs to be migrated to the current version, it is migrated.

Replace the project WAR file of a deployed runtime environment
Use this task to update the WAR files and libraries of the Worklight administration components. For example, apply a fix pack to the installation directory of Worklight Server.

Replace the WAR file of a deployed Worklight Server Configuration
>   If you applied a fix pack to your installation of Worklight Server, use this
>   task to update the console and administration WAR files of a deployed
>   configuration.

Other operations available in the Server Configuration Tool

Export a Configuration
>   When you click **File** > **Export Configuration as Ant files**, Ant files are
>   exported. These Ant files contain tasks that take the following actions for
>   the Worklight Console and Administration Services, and for each Worklight
>   runtime environment of the configuration:
>   - 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, 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 interface).
>
>   If you modify the Worklight Server status outside the Server Configuration
>   Tool, the status for this configuration is no longer accurate.

Migrate a V6.1.0 Configuration
>   Worklight configurations that were created with IBM Worklight V6.1.0 are
>   displayed in a folder called **Worklight 6.1 Configurations**. To migrate such
>   configurations to IBM Worklight Foundation V6.2.0, complete the following
>   steps:
>   1. Select the configuration.
>   2. Right-click to open a contextual menu.
>   3. Select **Migrate a V6.1 configuration**. An IBM Worklight Foundation
>   V6.2.0 configuration is created.
>   4. Review all the pages of the wizard. In Database Additional Settings,
>   you must enter information for the new administration database.
>   5. When all the pages are reviewed, click **Migrate**.
>
>   The Worklight runtime environment for V6.1.0 is removed from the
>   application server. The databases are migrated, the Worklight Console and
>   Administration services are deployed, and the Worklight runtime
>   environment is deployed.

Change the working directory where the configurations are stored
>   Click **File** > **Preferences** and select a different 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 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 Foundation" on page 13.

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.

To deploy a project WAR file and configure your application server, use the Ant tasks in "Deploying a project WAR file and configuring the application server with Ant tasks" on page 1141.

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 Foundation. 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.

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the DB2 databases manually" on page 1121 instead.

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 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 IBM Worklight Foundation 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.

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the MySQL databases manually" on page 1131 instead.

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 Foundation runs.

Creating the Oracle databases:

This section explains the procedures used to create the Oracle databases.

Before you begin

You perform this procedure to create the databases before you run Ant tasks or the Server Configuration Tool to populate them. For a fully manual database installation, see "Configuring the Oracle databases manually" on page 1136 instead.

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 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 system privilege: **CREATE SESSION**
           - Add system privilege: **CREATE SEQUENCE**
           - Add system privilege: **CREATE TABLE**
           - Add system privilege: **UNLIMITED TABLESPACE**

     b. Repeat step "a" to create a user, for example, named *WORKLIGHTREPORTS* for the 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 CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WORKLIGHT;
     DISCONNECT;

     CONNECT system/<system_password>@ORCL
     CREATE USER WORKLIGHTREPORTS IDENTIFIED BY 'WORKLIGHTREPORTS_password';
     GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WORKLIGHTREPORT
     DISCONNECT;
     ```

Creating and configuring the databases with Ant tasks:

Use Ant tasks to create and configure the 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 Foundation.
- If the Worklight table was created for IBM Worklight V6.1.0 or before, migrate the administration data to the administration database. In this case, the element **adminDatabase** is required in element **configuredatabase** for kind **'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 "Sample configuration files" on page 1156.

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 1110 must be completed.

Ant **configuredatabase** *task reference:*

Reference information for the **configuredatabase** Ant task.

Overview

The **configuredatabase** Ant task creates the databases that are used by Worklight Administration Services and by the Worklight runtime. This Ant task configures a database for a Worklight Project through the following actions:

- Checks whether the Worklight tables exist and creates them if necessary.
- If the tables exist for an older version of Worklight, migrates them to the current version.
- If the tables exist for the current version of Worklight, does nothing.

In addition, if the inner element <dba> is present, the task can have the following effects:

- Create the database if necessary.
- Create a user, if necessary, and grants that user access rights to the database.

IBM Worklight Foundation V6.2.0 introduces a new database that is referenced with kind **WorklightAdmin** for Worklight Administration Services. This database can support one Worklight runtime or more, and can handle the artifacts of those Worklight runtimes.

Important: If you upgrade from a Worklight version earlier that V6.2.0, you must also migrate the data from the Worklight runtime to the new database for Worklight Administration Services. IBM Worklight Foundation V6.2.0 introduces a new element, admindatabase, for this purpose, as shown in Table 2.

Attributes and elements for **configuredatabase**

The **configuredatabase** task has the following attributes:

Table 165. Attributes for the **configuredatabase** *Ant task*

Attribute	Description	Required	Default
kind	Type of database: **Worklight**, **WorklightReports**, or **WorklightAdmin**	Yes	None

IBM Worklight Foundation V6.2.0 supports three kinds of database: Worklight runtimes use **Worklight** and **WorklightReports** databases. Worklight Administration Services use the **WorklightAdmin** database.

The configuredatabase task supports the following elements:

Table 166. Inner elements for the configuredatabase *Ant task*

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

Table 166. Inner elements for the `configuredatabase` Ant task (continued)

Element	Description	Count
`admindatabase`	Parameters for migrating data from IBM Worklight V6.1.x runtime to IBM Worklight Foundation V6.2.0 Administration Services database	0..1

For each database type, you can use a `property` element to specify a JDBC connection property for access to the database. The <property> element has the following attributes:

Table 167. Attributes for the `property` element

Attribute	Description	Required	Default
`name`	Name of the property	Yes	None
`value`	Value for the property	Yes	None

Attributes and elements for `admindatabase`

Use the <admindatabase> element for migrating data from a Worklight runtime database to the Worklight Administration Services database. This element is mandatory when you migrate your Worklight runtime projects from V6.1.x and the `kind` attribute of **`configuredatabase`** is **`Worklight`**.

The `admindatabase` element has the following attribute.

Table 168. Attribute for the `admindatabase` element

Attribute	Description	Required	Default
`runtimeContextRoot`	Context root of the Worklight runtime	Yes	None

Because the Worklight Administration Services can handle one or more Worklight runtimes, you must reference a specific context root for each runtime. Use the `runtimeContextRoot` attribute to specify this context root. After Worklight data is migrated, you cannot change the context root of the Worklight runtime, unless the Worklight Administration Services database is removed and a new database is created.

The <admindatabase> element supports the following elements.

Table 169. Inner elements for the `admindatabase` element

Element	Description	Count
`derby`	Parameters for Derby	0..1
`db2`	Parameters for DB2	0..1
`driverclasspath`	JDBC driver class path	0..1
`mysql`	Parameters for MySQL	0..1
`oracle`	Parameters for Oracle	0..1

Apache Derby

The derby element has the following attributes:

Table 170. Attributes for the *derby* element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT*, *WLREPORT*, or *WLADMIN*, depending on kind.
datadir	Directory that contains the databases	Yes	None
schema	Schema name	No	*WORKLIGHT*, *WORKLIGHT*, or *WLADMINISTRATOR*, depending on kind

The derby element supports the following elements:

Table 171. Inner elements for the *derby* element

Element	Description	Count
property	JDBC connection property	0..∞

For the available properties, see Setting attributes for the database connection URL.

DB2

The db2 element has the following attributes:

Table 172. Attributes for the *db2* element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT*, *WLREPORT*, or *WLADMIN*, depending on kind
server	Host name of the database server	Yes	None
port	Port on the database server	No	50000
user	User name for accessing databases	Yes	None
password	Password for accessing databases	No	Queried interactively
instance	Name of the DB2 instance	No	Depends on the server
schema	Schema name	No	Depends on the user

For more information about DB2 user accounts, see DB2 security model overview.

The db2 element supports the following elements:

Table 173. Inner elements for the *db2* element

Element	Description	Count
property	JDBC connection property	0..∞

Table 173. Inner elements for the db2 element (continued)

Element	Description	Count
dba	Database administrator credentials	0..1

For the available properties, see Properties for the IBM Data Server Driver for JDBC and SQLJ.

The inner element <dba> specifies credentials for database administrators. This element has the following attributes:

Table 174. 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 the SYSADM or SYSCTRL DB2 privilege. For more information, see Authorities overview.

The <driverclasspath> element must contain JAR files for the DB2 JDBC driver and for the associated license. You can retrieve those files in one of the following ways:

- Download DB2 JDBC drivers from the DB2 JDBC Driver Versions page
- Or fetch the db2jcc4.jar file and its associated db2jcc_license_*.jar files from the DB2_INSTALL_DIR/java directory on the DB2 server.

You cannot specify details of table allocations, such as the table space, by using the Ant task. To control the table space, you must use the manual instructions in section "Configuring the DB2 databases manually" on page 1121.

MySQL

The element mysql has the following attributes:

Table 175. Attributes for the mysql element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT, WLREPORT,* or *WLADMIN,* depending on kind

server	Host name of the database server	Yes	None

port	Port on the database server	No	3306

user	User name for accessing databases	Yes	None

password	Password for accessing databases	No	Queried interactively

For more information about MySQL user accounts, see MySQL User Account Management.

The `mysql` element supports the following elements:

Table 176. Inner elements for the `mysql` element

Element	Description	Count
`property`	JDBC connection property	0..∞
`dba`	Database administrator credentials	0..1
`client`	The host that is allowed to access the database	0..∞

For the available properties, see Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J.

The inner element `dba` specifies database administrator credentials. This element has the following attributes:

Table 177. Attributes for the `dba` element for MySQL databases

Attribute	Description	Required	Default
`user`	User name for accessing databases	Yes	None
`password`	Password for accessing databases	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.

Each `client` inner element 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 178. 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 information about the `hostname` syntax, see Specifying Account Names.

The `driverclasspath` element must contain a MySQL Connector/J JAR file. You can download that file from the Download Connector/J page.

Alternatively, you can use the `mysql` element with the following attributes:

Table 179. Alternative attributes for the `mysql` element

Attribute	Description	Required	Default
`url`	Database connection URL	Yes	None
`user`	User name for accessing databases	Yes	None
`password`	Password for accessing databases	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 or the user, nor does it attempt to grant access to the user. The **configuredatabase** task ensures only 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>.

Oracle

The element oracle has the following attributes:

Table 180. Attributes for the oracle element

Attribute	Description	Required	Default
database	Database name	No	*ORCL*
server	Host name of the database server	Yes	None
port	Port on the database server	No	1521
user	User name for accessing databases	Yes	None
password	Password for accessing databases	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 exist yet

For more information about Oracle user accounts, see Overview of Authentication Methods.

The oracle element supports the following elements:

Table 181. Inner elements for the oracle element

Element	Description	Count
property	JDBC connection property	0..∞
dba	Database administrator credentials	0..1

For information about the available connection properties, see Class OracleDriver.

The inner element dba specifies database administrator credentials. This element has the following attributes:

Table 182. Attributes for the dba element for Oracle databases

Attribute	Description	Required	Default
user	User name for accessing databases	Yes	None
password	Password for accessing databases	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 table allocation, such as the table space, by using the Ant task. To control the table space, you can create the user account manually and assign it a default table space before running the Ant task. To control other details, you must use the manual instructions in section "Configuring the Oracle databases manually" on page 1136.

Alternatively, you can use the `oracle` element with the following attributes:

Table 183. Alternative attributes for the `oracle` element

Attribute	Description	Required	Default
`url`	Database connection URL	Yes	None
`user`	User name for accessing databases	Yes	None
`password`	Password for accessing databases	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 task does not attempt to create the database or the user, nor does it attempt to grant access to the user. The **configuredatabase** task ensures only 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 Foundation 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 1110
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.

   Note: At this stage, you can choose to provide a database user with limited privileges to better secure access to the Application Server database during runtime operations. To create a database user with restricted privileges, see "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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 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** and **WLREPORT** databases that you previously created, and **password** after **password=** is this user's password. If you 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).

   Note: The database user that is provided in step 2 does not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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

   The jndiName attributes must depend on the context root that you select for the Worklight Server application, following the instructions in "Configuring the

WebSphere Liberty Profile manually" on page 1164. 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. 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/*project-name*/db2.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/*project-name*/db2.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*project-name*/db2.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*project-name*/db2.
   - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/*project-name*/db2.

   If the directory for the JDBC driver JAR file does not exist, you must 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) to the directory that you determined in step 1.
3. Set up the JDBC provider:
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the appropriate scope from the **Scope** combination box.
   c. Click **New**.
   d. Set **Database type** to **DB2**.
   e. Set **Provider type** to **DB2 Using IBM JCC Driver**.
   f. Set **Implementation Type** to **Connection pool data source**.
   g. Set **Name** to **DB2 Using IBM JCC Driver**.
   h. Click **Next**.
   i. Set the **Class path** to the set of JAR files in the directory that you determined in step 1, one per line, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.
   j. Do not set **Native library path**.
   k. Click **Next**.
   l. Click **Finish**.

m. The JDBC provider is created.

n. Click **Save**.

4. Create a data source for the Worklight database:

a. Select **Resources** > **JDBC** > **Data Sources**.

b. Select the appropriate scope from the **Scope** combination box.

c. Click **New** to create a data source.

d. Set the **Data source name** to **Worklight Database**.

e. Set **JNDI Name** to **jdbc/WorklightDS**.

f. Click **Next**.

g. 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)** selected.

h. Click **Next**.

i. Create **JAAS-J2C** authentication data, specifying the DB2 user name and password as its properties. If necessary, go back to the data source creation wizard, by repeating steps 4a to 4h.

j. Select the authentication alias that you created in the **Component-managed authentication alias** combination box (not in the **Container-managed authentication alias** combination box).

k. Click **Next** and **Finish**.

l. Click **Save**.

m. In **Resources** > **JDBC** > **Data sources**, select the new data source.

n. Click **WebSphere Application Server data source properties**.

o. Select the **Non-transactional data source** check box.

p. Click **OK**.

q. Click **Save**.

r. Click **Custom properties** for the data source, select property **currentSchema**, and set the value to the schema used to create the data source tables (WRKSCHM and WLRESCHM in this example).

5. Create a data source for the Worklight reports database:

a. Select **Resources** > **JDBC** > **Data Sources**.

b. Select the appropriate scope from the **Scope** combination box.

c. Click **New** to create a data source.

d. Set the **Data source name** to **Worklight Reports Database**.

e. Set **JNDI Name** to **jdbc/WorklightReportsDS**.

f. Click **Next**.

g. Enter properties for the data source, for example:

- **Driver type**: 4
- **Database Name**: WLREPORT
- **Server name**: localhost
- **Port number**: 50000 (default)

Leave **Use this data source in (CMP)** selected.

h. Click **Next**.

i. If you need a different DB2 user name and password for the reports database than for the runtime database, create **JAAS-J2C** authentication data, specifying the DB2 user name and password as its properties. If necessary, go back to the data source creation wizard, by repeating steps 5a on page 1125 to 5h on page 1125.

j. Select the authentication alias that you created in the **Component-managed authentication alias** combination box (not in the **Container-managed authentication alias** combination box).

k. Click **Next** and **Finish**.

l. Click **Save**.

m. In **Resources** > **JDBC** > **Data sources**, select the new data source.

n. Click **WebSphere Application Server data source properties**.

o. Select the **Non-transactional data source** check box.

p. Click **OK**.

q. Click **Save**.

r. 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).

6. Test the data source connection by selecting each **Data Source** and clicking **Test Connection**.

7. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

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://db2server: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://db2server:50000/WLREPORT:currentSchema=WLRESCHM;"/>
  ...
</Context>
```

Where **worklight** after **username=** is the name of the system user with "CONNECT" access to the **WRKLGHT** and **WLREPORT** databases that you 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. Also, replace **db2server** with the host name of your DB2 server (for example, localhost, if it is on the same machine).

Note: The database user that is provided in this step does not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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

Configuring the Apache Derby databases manually:

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

Procedure
1. Create the databases and the tables within them. This step is described in "Setting up your Apache Derby databases manually"
2. Configure the application server to use this database setup. Go to one of the following topics:
   - "Configuring Liberty Profile for Derby manually" on page 1128
   - "Configuring WebSphere Application Server for Derby manually" on page 1128
   - "Configuring Apache Tomcat for Derby manually" on page 1131

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="10
  <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 1164. 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. 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/*project-name*/derby.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/ *project-name*/derby.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*project-name*/derby.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*project-name*/derby.
   - For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/*project-name*/derby.

   If the directory for the JDBC driver JAR file does not exist, you must create it.

2. Add the Derby JAR file from WORKLIGHT_INSTALL_DIR/ApplicationCenter/tools/ lib/derby.jar to the directory that you determined in step 1.

3. Set up the JDBC provider.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.
   b. Select the appropriate scope from the **Scope** combination box.
   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 that you 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 Worklight database.
   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources**.
   b. Select the appropriate scope from the **Scope** combination box.
   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).

5. Set up the data source for the Worklight reports database.

   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **Data sources**.

   b. Select the appropriate scope from the **Scope** combination box.

   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.

6. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

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 1111
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 Foundation runs.

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

   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:

   ```xml
   <!-- 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. If you have defined either a different user name, or a different password, or both, replace these entries accordingly. Also, replace **mysqlserver** with the host name of your MySQL server (for example, localhost, if it is on the same machine).

Note: The database user that is provided in this step does not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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 1164. 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. 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/*project-name*/mysql.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/*project-name*/mysql.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*project-name*/mysql.
   - For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*project-name*/mysql.

- For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/project-name/mysql.

If the directory for the JDBC driver JAR file does not exist, you must create it.

2. Add the MySQL JDBC driver JAR file that you downloaded from Download Connector/J 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. Select the appropriate scope from the **Scope** combination box.

   c. Click **New**.

   d. Create a **JDBC provider** named **MySQL**.

   e. Set **Database type** to **User defined**.

   f. Set **Scope** to **Cell**.

   g. Set **Implementation class** to **com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource**.

   h. Set the **Class path** to the JAR file in the directory that you determined in step 1, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.

   i. Save your changes.

4. Create a data source for the Worklight database:

   a. Click **Resources** > **JDBC** > **Data sources**.

   b. Select the appropriate scope from the **Scope** combination box.

   c. Click **New** to create a data source.

   d. Type any name (for example, Worklight Database).

   e. Set **JNDI Name** to jdbc/WorklightDS.

   f. Use the existing **JDBC Provider MySQL**, defined in the previous step.

   g. Set Scope to **New**.

   h. On the **Configuration** tab, select **Non-transactional data source**.

   i. Click **Next** a number of times, leaving all other settings as defaults.

   j. Save your changes.

5. Create a data source for the Worklight reports database:

   a. Click **Resources** > **JDBC** > **Data sources**.

   b. Select the appropriate scope from the **Scope** combination box.

   c. Click **New** to create a data source.

   d. Type any name (for example, Worklight Report Database).

   e. Set **JNDI Name** to jdbc/WorklightReportsDS.

   f. Use the existing **JDBC Provider MySQL**, defined in the previous step.

   g. Set Scope to **New**.

   h. On the **Configuration** tab, select **Non-transactional data source**. **New**.

   i. Click **Next** a number of times, leaving all other settings as defaults.

   j. Save your changes.

6. 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
```

> **Note:** The database user that is listed in this step does not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

7. 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**.

8. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

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://mysqlserver: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://mysqlserver:3306/WLREPORT"/>
  ...
</Context>
```

Where **worklight** after **username=** is the user name of the MySQL server, **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).

Note: The database user that is listed in this step does not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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 1112
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 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 system privilege: CREATE SESSION
      ```

```
            Add system privilege: CREATE SEQUENCE
            Add system privilege: CREATE TABLE
            Add system privilege: UNLIMITED TABLESPACE
```

Repeat the previous step to create the user WORKLIGHTREPORTS for the
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 CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WORKLIGHT;
DISCONNECT;
CONNECT system/system_password@ORCL
CREATE USER WORKLIGHTREPORTS IDENTIFIED BY WORKLIGHTREPORTS_password;
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE, UNLIMITED TABLESPACE TO WORKLIGHTREPORTS;
DISCONNECT;
```

3. Create the database tables for the Worklight database and Worklight reports
   database:

   a. Using the Oracle SQLPlus command-line interpreter, create the required
      tables for the 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 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 **WORKLIGHT** and **WORKLIGHTREPORTS** after **user=** are the names of the users with "CONNECT" access to the **WRKLGHT** and **WLREPORT** databases that you previously created, and **password** after **password=** are this users' passwords. If you defined either different user names, or different passwords, or both, replace these entries accordingly. Also, replace **oserver** with the host name of your Oracle server (for example, localhost, if it is on the same machine).

Note: The database users that are provided in this step do not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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 1164. If the context root is /app_context, use jndiName="app_context/jdbc/WorklightDS" and jndiName="app_context/jdbc/WorklightReportsDS".

Configuring WebSphere Application Server for Oracle manually:

If you want to manually set up and configure your Oracle database 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/*project-name*/oracle.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/*project-name*/oracle.
   - For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*project-name*/oracle.

- For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*project-name*/oracle.
- For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/*project-name*/oracle.

If the directory for the JDBC driver JAR file does not exist, you must create it.

2. Add the Oracle ⌂ojdbc6.jar file downloaded from JDBC and Universal Connection Pool (UCP) to the directory that you determined in step 1.

3. Set up the JDBC provider:

   a. In the WebSphere Application Server console, click **Resources** > **JDBC** > **JDBC Providers**.

   b. Select the appropriate scope from the **Scope** combination box.

   c. Click **New**.

   d. Complete the JDBC Provider fields as indicated in the following table:

Table 184. JDBC Provider field values

Field	Value
Database type	Oracle
Provider type	Oracle JDBC Driver
Implementation type	Connection pool data source
Name	Oracle JDBC Driver

   e. Click Next.

   f. Set the **Class path** to the JAR file in the directory that you determined in step 1, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.

   g. Click **Next**.

   The JDBC provider is created.

4. Create a data source for the Worklight database:

   a. Click **Resources** > **JDBC** > **Data sources**.

   b. Select the appropriate scope from the **Scope** combination box.

   c. Click **New**.

   d. Set **Data source name** to **Oracle JDBC Driver DataSource**.

   e. Set **JNDI name** to jdbc/WorklightDS.

   f. Click **Next**.

   g. Click **Select an existing JDBC provider** and select **Oracle JDBC driver** from the list.

   h. Click **Next**.

   i. 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).

   j. Click **Next** twice.

   k. Click **Resources** > **JDBC** > **Data sources** > **Oracle JDBC Driver DataSource** > **Custom properties**.

   l. Set **oracleLogPackageName** to **oracle.jdbc.driver**.

   m. Set **user = WORKLIGHT**.

   n. Set **password = *WORKLIGHT_password*.

o. Click **OK** and save the changes.

p. In **Resources** > **JDBC** > **Data sources**, select the new data source.

q. Click **WebSphere Application Server data source properties**.

r. Select **Non-transactional data source**.

s. Click **OK**.

t. Click **Save**.

5. Create a data source for the 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.

   Note: The database users **WORKLIGHT** and **WORKLIGHTREPORTS** that are provided in this step do not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

6. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

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 **WORKLIGHT** and **WORKLIGHTREPORTS** after **username=** are the names of the users with "CONNECT" access to the **WRKLGHT** and **WLREPORT** databases that you previously created, and **password** after **password=** are this users' passwords. If you defined either different user names, or different passwords, or both, replace these entries accordingly. Also, replace **oserver** with the host name of your Oracle server (for example, localhost, if it is on the same machine).

Note: The database users that are provided in this step do not need extended privileges on the databases. If you need to implement restrictions on the database, you can set here a user that has the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.

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.

Prerequisites

Verify that the user who runs the Ant tasks has the privileges that are described in "File system prerequisites" on page 72.

The `<configureapplicationserver>` Ant task

You can use the `<configureapplicationserver>` Ant task to configure an application server for a Worklight project. 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 of kind **Worklight** and of kind **WorklightReports**. See "Creating and configuring the databases with Ant tasks" on page 1114 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 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>
      <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>
      <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>
      <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>
      <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>
    <project warfile="/n/download/testWorklight.war" libraryfile="/n/download/worklight-jee-libr

    <!-- 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>
    <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 tasks for installation of Worklight runtime environments:

Reference information for the `configureapplicationserver`, `updateapplicationserver`, and `unconfigureapplicationserver` Ant tasks.

Task effects

`<configureapplicationserver>`

> The <configureapplicationserver> Ant task configures an application server to run a Worklight project WAR file as a web application. This task has the following effects.
>
> - It declares the Worklight web application in the specified context root, by default /worklight.
> - It deploys the project 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 file worklight-jee-library.jar and the database drivers in the application server.
> - It sets Worklight configuration properties through JNDI environment entries. These JNDI environment entries override the Worklight project default values that are contained in the worklight.properties file inside the WAR file.
> - On WebSphere Application Server, it configures a web container custom property.

`<updateapplicationserver>`

> The <updateapplicationserver> Ant task updates an already-configured Worklight web application on an application server. This task has the following effects.
>
> - It updates the project WAR file. The file must have the same base name as the project WAR file that was previously deployed.
> - It updates the Worklight Server runtime worklight-jee-library.jar library file.
>
> The task does not change the application server configuration, that is, the web application configuration, data sources, and JNDI environment entries.

<unconfigureapplicationserver>

> The <unconfigureapplicationserver> Ant task undoes the effects of an earlier <configureapplicationserver> run. This task has the following effects.
>
> - It removes the configuration of the Worklight web application with the specified context root. The task also removes the settings that have been added manually to that application.
> - It removes the project 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 `worklight-jee-library.jar` library file and the database drivers from the application server.
> - It removes the associated JNDI environment entries.

Attributes and elements

The <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> tasks have the following attributes:

Table 185. Attributes for the <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> Ant tasks

Attribute	Description	Required	Default
`contextroot`	Common prefix in URLs to the application (context root)	No	`/worklight`
`id`	Distinguishes different deployments	No	Empty
`environmentId`	Distinguishes different Worklight environments	No	Empty
`wasStartingWeight`	Start order for WebSphere Application Server. Lower values start first.	No	2
`shortcutsDir`	Directory where to place shortcuts	No	None

`contextroot` and `id`

> 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 compatibility with Worklight V5.x applications.
>
> In WebSphere Application Server Liberty profiles 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.

`environmentId`

> Use the `environmentId` attribute to distinguish several environments, consisting each of Worklight Server administration and Worklight runtime web applications, that must operate independently. You must set this attribute to the same value for the runtime application as the one that was set in the <installworklightadmin> invocation, for the Administration Services application.

`wasStartingWeight`

> Use the `wasStartingWeight` attribute to specify a value that is used in WebSphere Application Server as a weight to ensure that a start order is

respected. As a result of the start order value, the Worklight Administration Services web application is deployed and started before any other Worklight runtime projects. If Worklight projects are deployed or started before the web application, the JMX communication is not established and you cannot manage your Worklight projects.

shortcutsDir

The `shortcutsDir` attribute for the `<unconfigureApplicationServer>` Ant task specifies where to expect the shortcuts to the Worklight Console if it was installed by a version of the `<configureApplicationServer>` Ant task older than 6.2.0. If you set this attribute, the Ant task might remove the following files from that directory: `worklight-console.url`, `worklight-console.sh`, and `worklight-console.html`.

The `<configureapplicationserver>`, `<updateapplicationserver>`, and `<unconfigureapplicationserver>` tasks support the following elements:

Table 186. 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 `<project>` element specifies details about the project to deploy to the application server. It has the following attributes:

Table 187. 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 automigrate the WAR file to the current Worklight Server version	No	True
`migratedWarBackupFile`	Where to store a backup of the migrated WAR file	No	None

To create the `warfile` attribute, run the `<war-builder>` Ant task. See "Building a project WAR file with Ant" on page 1104.

By default, the WAR file is automatically migrated to the current Worklight Server version. In this case, you can request a backup of the migrated WAR file on disk before it is deployed in the application server. To do so, specify a value for the **migratedWarBackupFile** attribute. If you set the `migrate` attribute to `false`, the WAR file is not migrated and, if the Worklight version that produced the WAR file is not suitable for the Worklight Server version, the deployment fails.

The <property> element specifies a deployment property to be defined in the application server. It has the following attributes:

Table 188. Attributes for the <property> element

Attribute	Description	Required	Default value
`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 you can set, see "Configuration of Worklight applications on the server" on page 1171.

The <applicationserver> element describes the application server to which the Worklight application is deployed. It is a container for one of the following elements:

Table 189. 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 <websphereapplicationserver> element (or <was> in its short form) denotes a WebSphere Application Server instance, version 7.0 or newer. WebSphere Application Server full profile (Base, and Network Deployment) are supported, as is Liberty profile (Core). Liberty profile Network Deployment is not yet supported. The <websphereapplicationserver> element has the following attributes:

Table 190. 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 191. Inner elements for the <was> element (single-server deployment)

Element	Description	Count
`server`	A single server	0..1

The <server> element, which is used in this context, has the following attributes:

Table 192. 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 193. Inner elements for the <was> element (network deployment)

Element	Description	Count
cell	The entire cell	0..1

Element	Description	Count
cluster	All servers of a cluster	0..1

Element	Description	Count
node	All servers in a node, clusters excluded	0..1

Element	Description	Count
server	A single server	0..1

The <cell> element has no attributes.

The <cluster> element has the following attributes:

Table 194. Attributes for the <cluster> element (network deployment)

Attribute	Description	Required	Default
name	Cluster name	Yes	None

The <node> element has the following attributes:

Table 195. Attributes for the <node> element (network deployment)

Attribute	Description	Required	Default
name	Node name	Yes	None

The <server> element, which is used in a Network Deployment context, has the following attributes:

Table 196. Attributes for the <server> element (network deployment)

Attribute	Description	Required	Default
nodeName	Node name	Yes	None

Attribute	Description	Required	Default
serverName	Server name	Yes	None

The <tomcat> element denotes an Apache Tomcat server. It has the following attributes:

Table 197. 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, specify the value of the **CATALINA_BASE** environment variable.	Yes	None

The <reports> element specifies what set of BIRT *.rptdesign report files to instantiate for access to the database of Worklight reports.

The <reports> element has the following attribute:

Table 198. Attributes of the <reports> element

Attribute	Description	Required	Default
`todir`	Destination directory	Yes	None

The <reports> element supports the following element:

Table 199. 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 the report templates that are provided in the WorklightServer/report-templates/ directory in the Worklight Server distribution.

The <database> element specifies what information is necessary to access a particular database. Two databases must be declared: <database kind="Worklight"> and <database kind="WorklightReports">. The <database> element is specified like the <configuredatabase> Ant task, except that it does not have the <dba> and <client> elements. It might, however, have <property> elements. The<database> element has the following attributes:

Table 200. Attributes of the <database> element

Attribute	Description	Required	Default
`kind`	The kind of database: **Worklight** or **WorklightReports**	Yes	None

The<database> element supports the following elements:

Table 201. 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 <derby> element has the following attributes:

Table 202. Attributes of the <derby> element

Attribute	Description	Required	Default
`database`	Database name	No	**WRKLGHT** or **WLREPORT**, depending on the kind

| `datadir` | Directory that contains the databases | Yes | None |

| `schema` | Schema name | No | **WORKLIGHT** |

The <derby> element supports the following element:

Table 203. Inner element for the <derby> element

Element	Description	Count
`property`	Data source property or JDBC connection property	0..s∞

For more information about the available properties, see the documentation for Class EmbeddedDataSource40. See also the documentation for Class EmbeddedConnectionPoolDataSource40.

For more information about the available properties for a Liberty server, see the documentation for **properties.derby.embedded** at Liberty profile: Configuration elements in the server.xml file.

When the `worklight-ant-deployer.jar` file is used within the installation directory of Worklight, a `<driverclasspath>` element is not necessary.

To specify a DB2 database

The <db2> element has the following attributes:

Table 204. Attributes of the <db2> element

Attribute	Description	Required	Default
`database`	Database name	No	**WRKLGHT** or **WLREPORT**, depending on the kind

| `server` | Host name of the database server | Yes | None |

| `port` | Port on the database server | No | 50000 |

user	User name for accessing databases. This user does not need extended privileges on the databases. If you implement restrictions on the database, you can set a user with the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.	Yes	None
password	Password for accessing databases	No	Queried interactively
schema	Schema name	No	Depends on the user

For more information about DB2 user accounts, see DB2 security model overview.

The <db2> element supports the following element:

Table 205. Inner elements for the <db2> element

Element	Description	Count
property	Data source property or JDBC connection property	0..∞

For more information about the available properties, see Properties for the IBM Data Server Driver for JDBC and SQLJ.

For more information about 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 JAR files for the DB2 JDBC driver and the associated license. You can download DB2 JDBC drivers from DB2 JDBC Driver Versions.

To specify a MySQL database

The <mysql> element has the following attributes:

Table 206. Attributes of the <mysql> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT* or *WLREPORT*, depending on kind
server	Host name of the database server	Yes	None
port	Port on the database server	No	3306

user	User name for accessing databases. This user does not need extended privileges on the databases. If you implement restrictions on the database, you can set a user with the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.	Yes	None
password	Password for accessing databases	No	Queried interactively

Instead of `database`, `server`, and `port`, you can also specify a URL. In this case, use the following attributes:

Table 207. Alternative elements for the <mysql> element

Attribute	Description	Required	Default
url	URL for connection to the database	Yes	None

user	User name for accessing databases. This user does not need extended privileges on the databases. If you implement restrictions on the database, you can set a user with the restricted privileges that are listed in"Restricting database user permissions for Worklight Server runtime operations" on page 75.	Yes	None
password	Password for accessing databases	No	Queried interactively

For more information about MySQL user accounts, see MySQL User Account Management.

The `<mysql>` element supports the following element:

Table 208. Inner elements for the <mysql> element

Element	Description	Count
property	Data source property or JDBC connection property	0..∞

For more information about the available properties, see the documentation at Driver/Datasource Class Names, URL Syntax and Configuration Properties for Connector/J.

For more information about 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 `<oracle>` element has the following attributes:

Table 209. Attributes of the <oracle> element

Attribute	Description	Required	Default
database	Database name	No	*ORCL*
server	Host name of the database server	Yes	None
port	Port on the database server	No	1521
user	User name for accessing databases. This user does not need extended privileges on the databases. If you implement restrictions on the database, you can set a user with the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.	Yes	None
password	Password for accessing databases	No	Queried interactively

Instead of `database`, `server`, and `port`, you can also specify a URL. In this case, use the following attributes:

Table 210. Alternative attributes of the <oracle> element

Attribute	Description	Required	Default
url	URL for connection to the database	Yes	None
user	User name for accessing databases. This user does not need extended privileges on the databases. If you implement restrictions on the database, you can set a user with the restricted privileges that are listed in "Restricting database user permissions for Worklight Server runtime operations" on page 75.	Yes	None
password	Password for accessing databases	No	Queried interactively

For more information about Oracle user accounts, see Overview of Authentication Methods.

For more information about Oracle database connection URLs, see the **Database URLs and Database Specifiers** section at Data Sources and URLs.

It supports the following elements:

Table 211. Inner elements for the <oracle> element

Element	Description	Count
property	Data source property or JDBC connection property	0..∞

For more information about the available properties, see the **Data Sources and URLs** section at Data Sources and URLs.

For more information about 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 212. 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 values, usually `java.lang.String/Integer/Boolean`	No	`java.lang.String`
value	Value for the property	Yes	None

Sample configuration files:

IBM Worklight Foundation includes a number of sample configuration files to help you get started with the Ant tasks to install the Worklight Server administration and the Worklight runtime environment.

The easiest way to get started with the <configuredatabase>, <installworklightadmin> and <configureapplicationserver> 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 213. Sample configuration files provided with IBM Worklight Foundation

Task	Derby	DB2	MySQL	Oracle
Create databases with database administrator credentials	create-database-derby.xml	create-database-db2.xml	create-database-mysql.xml	create-database-oracle.xml

Table 213. Sample configuration files provided with IBM Worklight Foundation (continued)

Task	Derby	DB2	MySQL	Oracle
Install Worklight Server administration and Worklight runtime environment on Liberty	`configure-liberty-derby.xml`	`configure-liberty-db2.xml`	`configure-liberty-mysql.xml` (See Note)	`configure-liberty-oracle.xml`
Install Worklight Server administration and Worklight runtime environment on 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`

Install Worklight Server administration and Worklight runtime environment on WebSphere Application Server Network Deployment	`configure-wasnd-cluster-derby.xml`
	
	`configure-wasnd-server-derby.xml`
	
	`configure-wasnd-node-derby.xml`
	
	`configure-wasnd-cell-derby.xml`
		`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`
				`configure-wasnd-cluster-oracle.xml`
				
				`configure-wasnd-server-oracle.xml`
				
				`configure-wasnd-node-oracle.xml`
				
				`configure-wasnd-cell-oracle.xml`

Install Worklight Server administration and Worklight runtime environment on Apache Tomcat	`configure-tomcat-derby.xml`	`configure-tomcat-db2.xml`	`configure-tomcat-mysql.xml`	`configure-tomcat-oracle.xml`
Install or upgrade IBM Worklight V5.0.6 on Liberty	`redeploy506-liberty-derby.xml`	`redeploy506-liberty-db2.xml`	`redeploy506-liberty-mysql.xml` (See Note)	`redeploy506-liberty-oracle.xml`
Install or upgrade IBM Worklight V5.0.6 on WebSphere Application Server full profile, single server	`redeploy506-was-derby.xml`	`redeploy506-was-db2.xml`	`redeploy506-was-mysql.xml` (See Note)	`redeploy506-was-oracle.xml`

Table 213. Sample configuration files provided with IBM Worklight Foundation (continued)

Task	Derby	DB2	MySQL	Oracle

Install or upgrade IBM Worklight V5.0.6 on WebSphere Application Server Network Deployment	`redeploy506-wasnd-cluster-derby.xml`
	
	`redeploy506-wasnd-server-derby.xml`
	
	`redeploy506-wasnd-node-derby.xml`
	
	`redeploy506-wasnd-cell-derby.xml`
		`redeploy506-wasnd-cluster-db2.xml`
		
		`redeploy506-wasnd-server-db2.xml`
		
		`redeploy506-wasnd-node-db2.xml`
		
		`redeploy506-wasnd-cell-db2.xml`
			`redeploy506-wasnd-cluster-mysql.xml (See Note)`
			
			`redeploy506-wasnd-server-mysql.xml (See Note)`
			
			`redeploy506-wasnd-node-mysql.xml (See Note)`
			
			`redeploy506-wasnd-cell-mysql.xml`
				`redeploy506-wasnd-cluster-oracle.xml`
				
				`redeploy506-wasnd-server-oracle.xml`
				
				`redeploy506-wasnd-node-oracle.xml`
				
				`redeploy506-wasnd-cell-oracle.xml`

Install or upgrade IBM Worklight V5.0.6 on Apache Tomcat	`redeploy506-tomcat-derby.xml`	`redeploy506-tomcat-db2.xml`	`redeploy506-tomcat-mysql.xml`	`redeploy506-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 that is 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 WebSphere Application Server 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 `&`, unless you explicitly want to reference an XML entity.
- Double quotation marks (") must be written as `"`, except when inside a string that is 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 Worklight applications on the server" on page 1171. In production, you must often define the following specific properties:
- `publicWorkLightHostname`
- `publicWorkLightProtocol`
- `publicWorkLightPort`

Step 6

Run the commands:
```
ant -f configure-file.xml admdatabases
ant -f configure-file.xml databases
```

These commands ensure that the designated databases exist and contain the required tables for IBM Worklight Foundation. The target `admdatabases` must be run before the target `databases`. This is especially important in the case of an upgrade where management data is migrated from the Worklight database to the administration database, which needs to have been initialized first by the target `admdatabases`.

For an initial installation, and if a DBA has not created the databases manually, use the file in row "Create databases with database administrator credentials" of Table 213 on page 1156. These files add special parameters to the `configuredatabase` Ant task (the DBA credentials). The parameters enable the Ant task to create a database and a user if required.

Step 7

Run the command:
```
ant -f configure-file.xml adminstall
```

This command installs your Administration Services and Worklight Console components onto the application server.

To install updated Administration Services and Worklight Console components (for example, to apply a Worklight Server fix pack), run the command:

```
ant -f configure-file.xml minimal-admupdate
```

To reverse the installation step, run the command:
```
ant -f configure-file.xml admuninstall
```

This command uninstalls the Administration Services and Worklight Console components.

Step 8

Run the command:
```
ant -f configure-file.xml install
```

This command installs your Worklight runtime environment as a `.war` file onto the application server.

To install an updated Worklight runtime environment onto the application server, run the command:
```
ant -f configure-file.xml minimal-update
```

To reverse the installation step, run the command:
```
ant -f configure-file.xml uninstall
```

This command uninstalls the Worklight runtime environment.

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 administrative console to update the `.war` file, all properties that are configured for this web application are lost.

Configuring multiple Worklight projects in different environments:

Different environments operate independently. For example, with this approach you can host a test environment, a pre-production environment, and a production environment on the same server or in the same WebSphere Application Server Network Deployment cell.

You can configure multiple projects either with the same Administration Services and Worklight Console, or with separate Worklight environments. An environment consists of the following web applications:
• The Administration Services web application,
• Optionally, the Worklight Console web application,
• Some Worklight runtime environments or project web applications, each with the appropriate database schemas.

You specify the environment through the attribute `environmentId` in the `<installworklightadmin>` and `<configureapplicationserver>` invocations.

If you use this approach, you must respect the following constraints:
• Each Administration Services application must use a different administration database or schema.

- Each configuration for a Worklight runtime environment must use a different Worklight database or schema, and its own Worklight reports database or schema.
- Each configuration for a Worklight runtime environment must be deployed with the same `environmentId` attribute as the corresponding Administration Services.
- If the application server is WebSphere Application Server Liberty Profile, each 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 Worklight project must use either a different `environmentId` attribute or 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 Apache Tomcat, each 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 multiple Worklight projects in a single environment:

There are two ways to configure multiple projects on a single server. In a single environment, your Worklight projects are managed by the same Worklight Server administration components.

You can configure multiple projects either with the same Administration Services and Worklight Console, or with separate Worklight environments. An environment consists of the following web applications:

- The Administration Services web application,
- Optionally, the Worklight Console web application,
- Some Worklight runtime environments or project web applications, each with the appropriate database schemas.

It is possible to install different Worklight runtime components in the same application server, and have them managed by the same Administration Services and Worklight Console.

This configuration is possible even for Worklight projects that are built by using different versions of Worklight Studio. However, the projects must be migrated to the version of Worklight Server that is used to install the Worklight Server administration components. For more information about migrating projects, see "Migrating a project WAR file for use with a new Worklight Server" on page 1164.

If you use this configuration, you must respect the following constraints:

- Each Worklight project configuration must use a different Worklight database or schema, and each must use its own Worklight reports database or schema.
- If the application server is WebSphere Application Server Liberty Profile, each 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 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 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 you configure 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 Foundation 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 Runtime Component to be installed. The following table lists the available elements:

   Note: You must choose the same inner element to install the Worklight Administration Services. You must install an instance of the Worklight Administration Service on each server where the Worklight Runtime Component is installed.

Table 214. 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 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 administrative 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 215. 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.

Before you begin

Review the environment IDs. Specifying an environment ID is optional, but if it is defined, the value must be the same for each Worklight runtime environment that is managed by the same Worklight Server administration component, and this value must match the environment ID that is used when the Worklight Server administration component is installed. The environment ID is defined as an

application JNDI variable prefixed by the context root of the application. See step 5 of this procedure to define the environment ID.

For more information about the **ibm.worklight.admin.environmentid** JNDI property, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

About this task

In addition to modifications for the databases that are described in "Creating and configuring the databases manually" on page 1121, 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 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\resource
     ```

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">
       <privateLibrary>
         <fileset dir="${shared.resource.dir}/lib" includes="worklight-jee-library.jar"/>
       </privateLibrary>
     </classloader>
   </application>

   <!-- Declare web container custom properties for the Worklight Server application. -->
   <webContainer invokeFlushAfterService="false"/>
   ```

   This declaration installs the Worklight Server application with the context root `/worklight`. If you want to assign a different context root `/app_context`, start the declaration with one of the following code snippets:

   ```
   <application id="app_context" name="app_context" location="worklight.war" type="war">
   ```

or:

```
<application id="worklight" name="worklight" location="worklight.war" context-root="/app_context"
```

For more information about the context root, see WebSphere Application Server documentation about Deploying application to the Liberty Profile

5. If the Worklight Server Administration component uses an environment ID, declare that environment ID for Worklight Server application:

```
<jndiEntry jndiName="worklight/ibm.worklight.admin.environmentid" value='"ValueOfEnvironmentID"'/
```

In the declaration:

- `worklight` is the context root of the Worklight Server application. If you choose another value in previous step, replace "worklight" by that value.
- `ValueOfEnvironmentID` should be replaced by the value that is used for the Worklight Server administration component

Configuring WebSphere Application Server manually:

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

Before you begin

Find the SOAP port of the deployment manager (WebSphere Application Server Network Deployment only)

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.

For WebSphere Application Server Network Deployment, find the SOAP port of the deployment manager:

1. Open the System Administration/Deployment manager.
2. In additional properties, open 'Ports'
3. Note the value of **SOAP_CONNECTOR_ADDRESS**. This value is needed to set the value of the **ibm.worklight.admin.jmx.dmgr.port** environment entry for the Worklight Administration Services.

Review the environment IDs

Specifying an environment ID is optional, but if it is defined, the value must be the same for each Worklight runtime environment that is managed by the same Worklight Server administration component, and this value must match the environment ID that is used when the Worklight Server administration component is installed. For more information about the **ibm.worklight.admin.environmentid** JNDI property, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

Procedure

1. Determine a suitable file name for the Worklight shared library in the WebSphere Application Server installation directory.
   - For a standalone server, you can use a file name such as *WAS_INSTALL_DIR*/optionalLibraries/IBM/*project-name*/worklight-jee-library.jar.
   - For deployment to a WebSphere Application Server Network Deployment cell, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/*project-name*/worklight-jee-library.jar.

- For deployment to a WebSphere Application Server Network Deployment cluster, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*project-name*/worklight-jee-library.jar.
- For deployment to a WebSphere Application Server Network Deployment node, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*project-name*/worklight-jee-library.jar.
- For deployment to a WebSphere Application Server Network Deployment server, use *WAS_INSTALL_DIR*/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/*project-name*/worklight-jee-library.jar.

    If the directory does not exist, you must create it.

2. Copy the file ${WORKLIGHT_INSTALL_DIR}/WorklightServer/worklight-jee-library.jar to the location that you determined in step 1.
3. Log on to the WebSphere Application Server administration console for your Worklight Server. The address is of the form http://*server*.com:9060/ibm/console, where *server* is the name of the server.
4. Create the 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, enter the file name that you determined in step 1, replacing *WAS_INSTALL_DIR*/profiles/*profile-name* with the WebSphere Application Server variable reference ${USER_INSTALL_ROOT}.
    g. In **Class Loading**, select the check box **Use an isolated class loader for this shared library**.
5. Create the Worklight JDBC data source and provider. See the instructions for the appropriate DBMS in "Creating and configuring the databases manually" on page 1121
6. Add a specific web container custom property.
    a. Click **Servers** > **Server Types** > **Application Servers**, and select the server used for Worklight.
    b. Click **Web Container Settings** > **Web container**.
    c. Click **Custom properties**.
    d. Click **New**.
    e. Enter the property values listed in the following table:

Table 216. 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**.
7. Install a 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 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. In **Map environment Entries for Web Module**, you can assign the JNDI variables according to your configuration.

- Assign to the variable *ibm.worklight.topology.platform* to `WAS`
- Assign to the variable *ibm.worklight.admin.jmx.connector* to `SOAP`
- If the environment ID is set for the Administration Services, set the variable *ibm.worklight.admin.environmentid* to the same value.
- On a stand-alone WebSphere Application Server, assign the value of *ibm.worklight.topology.clustermode* to `Standalone`
- On WebSphere Application Server Network Deployment, assign the value of *ibm.worklight.topology.clustermode* to `Cluster`, assign the value of *ibm.worklight.admin.jmx.dmgr.host* to the host name of the deployment manager, and assign the value of *ibm.worklight.admin.jmx.dmgr.port* to the SOAP port of the deployment manager.

n. Click **Finish**.

8. (Optional). As an alternative to step 6, you can map the shared libraries as follows:

a. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > **worklight_war**.

b. In the References section, click **Shared library references**.

c. Select **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.

9. Define the Startup behavior

a. Click **Applications** > **Application Types** > **WebSphere enterprise applications** > **worklight_war**.

b. Click **Startup behavior**.

c. In *Startup Order*, enter 2.

Note: The Worklight Administration service must already be available when the Worklight Runtime starts.

10. 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**.

11. 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**.

12. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

Results

You can now view the runtime component from the Worklight Administration Console that is installed in "Installing and configuring Worklight Server administration" on page 104. The default URL of the Worklight Administration Console is`http://<server>:<port>`/worklightconsole, 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.

Before you begin

Review the environment IDs. Specifying an environment ID is optional, but if it is defined, the value must be the same for each Worklight runtime environment that is managed by the same Worklight Server administration component, and this value must match the environment ID that is used when the Worklight Server administration component is installed. For more information about the `ibm.worklight.admin.environmentid` JNDI property, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

Procedure

1. Copy the 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 1121.

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 Runtime Component. -->
<Context path="/worklight" docBase="worklight">
    <Environment name="ibm.worklight.topology.platform" value="Tomcat" type="java.lang.String" o
    <Environment name="ibm.worklight.topology.clusterMode" value="Standalone" type="java.lang.St
</Context>
```

   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"`.

   If the environment ID is set for the Administration Services, set the variable *ibm.worklight.admin.environmentid* to the same value.

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 Worklight applications on the server

You can configure each Worklight application by specifying a set of configuration parameters on the server side.

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 a 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 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 a Worklight project in production by using JNDI environment entries" on page 1182 to learn about the JNDI environment entries that you can specify in a production environment. Properties that are relevant only in development environments are not available as JNDI entries.

Within the `worklight.properties` file, you can define some application-specific configuration properties; for example, to configure the URL of a service that is called from the mobile application and the URL is different in production, development, and test environments. See "Declaring and using application-specific configuration properties" on page 1179 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 in the following circumstances:

- You are using relative path for the **onLoginUrl** parameter in the `authenticationConfig.xml` file.

- You are generating the URL for mobile web and desktop browser apps from the console.

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 217. Worklight Server location properties

Property name	Description
publicWorkLightHostname	The IP address or host name of the computer running IBM Worklight Foundation.
	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 a Worklight project in production by using JNDI environment entries" on page 1182

For information about how to specify configuration properties, see "Configuration of Worklight applications on the server" on page 1171.

Worklight database setup for development mode

IBM Worklight Foundation uses defaults to access the 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 1121.

Property keys and values for JDBC-based properties

Property Key	Property Value
`wl.db.url`	JDBC path to Worklight database.
`wl.db.username`	Worklight database user name. Default: `Worklight`
`wl.db.password`	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 Worklight Reports database Default: refers to Worklight database
`wl.reports.db.username(*)`	Worklight Reports database user name. Default: refers to Worklight database
`wl.reports.db.password(*)`	Worklight Reports database password Default: refers to Worklight database

Note: (*) By default all Worklight report mechanisms use a single reports database. The reports database is set to be the same as the Worklight database. For information about how this default setting can be changed, see "Using raw data reports" on page 1424.

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.

The following properties are required only when a proxy is used to route requests to APNS, GCM, or SMS push servers. When no proxy is used, it is not necessary to set the properties (the `*.enabled` property value should be set to `false`).

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.

Note:
- For push notifications to work, you must ensure that your firewall accepts outgoing connections to `android.googleapis.com` on port 443.

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.
`push.apns.proxy.port`	APNS proxy port.
`push.apns.proxy.user`	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
`push.apns.proxy.password`	Proxy password, if the proxy requires authentication.

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 Foundation creates activity logs and sends them to a server for analysis.

You can modify how the Worklight Server forwards analytics data to the IBM Worklight Analytics Platform by editing the following properties files.

Table 218. IBM Worklight Analytics Platform properties.

Property Name	Default Value	Description
`wl.analytics.url`	None.	Required. The URL that is exposed by the IBM Worklight Analytics Platform that receives incoming analytics data. Example: `http://<hostname>:<port>/<context-root>/d`
`wl.analytics.logs.forward`	true	When the property is set to `true`, server logs that are recorded on the Worklight Server are captured and forwarded to the IBM Worklight Analytics Platform.
`wl.analytics.username`	None.	The user name that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
`wl.analytics.password`	None.	The password that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
`wl.analytics.queues`	20	Maximum number of concurrent queues that are used to buffer data before it is forwarded to the IBM Worklight Analytics Platform. For more information, see "Worklight throughput-tuning" on page 1415.
`wl.analytics.queue.size`	10	Size of each queue that is used to buffer data before it is forwarded to the IBM Worklight Analytics Platform. For more information, see "Worklight throughput-tuning" on page 1415.

Property Name	Default Value	Description
`wl.analytics.fowardInterval`	`5Seconds`	Time interval in seconds to trigger an automatic forward of buffered data to the IBM Worklight Analytics Platform. Set the property to -1 to disable the automatic forwarding. For more information, see "Worklight throughput-tuning" on page 1415.

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 Foundation 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 219. 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.
`ssl.websphere.config`	Set this property to `true` for HTTP adapters to use the WebSphere SSL configuration. Default: `false`.
`ssl.websphere.alias`	The alias of the WebSphere SSL configuration that HTTP adapters use

For descriptions of other Worklight configuration properties, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182

For information about how to specify Worklight configuration properties, see "Configuration of Worklight applications on the server" on page 1171.

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 738.

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 743.

  Note: The password that is specified in **ssl.keystore.password** is not the same password as the one 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**, **bitly.apikey**, **compress.response.threshold**, and **adapters.saxparser.doctype.validation** parameters.

Property keys and values for the **serverSessionTimeout**, **bitly.username**, **bitly.apikey**, **compress.response.threshold**, and **adapters.saxparser.doctype.validation** parameters.

Property Key	Property Value
serverSessionTimeout	Client inactivity timeout, after which the 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 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.
adapters.saxparser.doctype.validation	`True` or `False`. If set to `False`, the adapter does not validate the XML response received from the back-end server. This might be useful in cases where the time required to validate could be expected to exceed the allowed timeout value. The default setting is `True`, meaning that the server validates the response.

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 1179 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 Foundation 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 Foundation configuration is accessed for a property. If the property is not found, but the same encrypted property (with `.enc` suffix) is defined, IBM Worklight Foundation 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 Foundation 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 documentation.

Obsolete properties

Some properties are no longer required.

Table 220. 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`
Single identity login module security settings	`console.username`, `console.password`

Declaring and using application-specific configuration properties

Use the `${propertyName}` notation to reuse application-specific properties that are declared in the `worklight.properties` file.

As a developer, you might want to parameterize some elements in the configuration of the server side of your Worklight application so that an IT administrator can change the value in production. For example, a Worklight adapter might need to call a back-end service, and the URL of this service might be different in a production environment from its value in a development environment. In this scenario, you can create a new Worklight configuration property to store the URL, and the IT administrator can then set the final production value as a JNDI environment entry.

You can declare application-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 a Worklight project in production by using JNDI environment entries" on page 1182) 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 typeface) 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 881. For **loginModule** parameters, see "Configuring login modules" on page 900.

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 745.

Configuring a Worklight project in production by using JNDI environment entries

When you deploy a 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 full profile) or the `server.xml` file (WebSphere Application Server Liberty profile, Apache Tomcat).

About this task

Many of the 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 1178:

- You can define the property with the `.enc` suffix in the `worklight.properties` file that is packaged in the WAR file of the Worklight project. You can then override the encrypted value by using a JNDI property. With Apache Tomcat, this option is the only one available.
- On WebSphere Application Server full profile 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 Worklight properties that are always available as JNDI entries:

Table 221. Worklight properties available as JNDI entries

Property name	Description
`adapters.saxparser.doctype.validation`	Specifies whether the adapter should validate the XML response received from the back-end server. If set to `False`, the adapter does not validate the response. This might be useful in cases where the time required to validate could be expected to exceed the allowed timeout value. Default: `True`
`publicWorkLightHostname`	The IP address or host name of the computer that is running IBM Worklight Foundation. 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.
`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.

Table 221. Worklight properties available as JNDI entries (continued)

Property name	Description
`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.apns.proxy.user`	Proxy user name, if the proxy requires authentication. Empty user name means no authentication.
`push.apns.proxy.password`	Proxy password, if the proxy requires authentication.
`push.sms.proxy.protocol`	Push SMS proxy protocol.
`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`.
`ssl.websphere.config`	Set this property to `true` for HTTP adapters to use the WebSphere SSL configuration. Default: `false`.
`ssl.websphere.alias`	The alias of the WebSphere SSL configuration that HTTP adapters use
`cluster.data.synchronization.taskFrequencyInSeconds`	Applications and adapters cluster data synchronization interval. Default: 2.

Table 221. Worklight properties available as JNDI entries (continued)

Property name	Description
`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 Worklight production servers. It is not supported on the Worklight Studio development environment.
`wl.analytics.url`	The URL that is exposed by the IBM Worklight Analytics Platform that receives incoming analytics data. Example: `http://<hostname>:<port>/<context-root>/data`.
`wl.analytics.username`	The user name that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
`wl.analytics.password`	The password that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
`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.clientlogs.adapter.name`	The name of the HTTP adapter that you want to use to receive client-side logs. If you do not specify this property, the default `WLClientLogReceiver` name is used.

Table 221. Worklight properties available as JNDI entries (continued)

Property name	Description
`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.
`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.
`ibm.worklight.admin.rmi.registryPort`	Optional. RMI registry port for the JMX connection through a firewall. Tomcat only.
`ibm.worklight.admin.rmi.serverPort`	Optional. RMI server port for the JMX connection through a firewall. Tomcat only.
`ibm.worklight.admin.jmx.connector`	Mandatory. JMX connector type, by default RMI/SOAP. WebSphere Application Server profile profile only.
`ibm.worklight.admin.jmx.dmgr.host`	Mandatory. Deployment Manager host name. WebSphere Application Server Network Deployment only.
`ibm.worklight.admin.jmx.dmgr.port`	Mandatory. Deployment Manager RMI or SOAP port. WebSphere Application Server Network Deployment only.
`ibm.worklight.admin.environmentid`	Optional. Environment identifier for the registration of the MBeans. Use this identifier when different instances of the Worklight Server are installed on the same application server. The identifier determines which Administration Services, which console, and which runtimes belong to the same installation. The Administration Services manage only the runtimes that have the same environment identifier.

Table 221. Worklight properties available as JNDI entries (continued)

Property name	Description
`ibm.worklight.admin.serverid`	Optional. Server identifier. Must be different for each server in the farm. Farm of servers only.
`ibm.worklight.topology.platform`	Server type. The values can be: • `Liberty` • `WAS` • `Tomcat` If the default value is not set, the application tries to guess the server type.
`ibm.worklight.topology.clustermode`	In addition to the server type, you must specify the server topology. The values that are allowed: • `Standalone` • `Cluster` • `Farm` The default value is `Standalone`.

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 1141), 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 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 133

Figure 133. Setting JNDI environment entries on WebSphere Application Server

Related reference:

"Configuration of Worklight applications on the server" on page 1171
You can configure each 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 Foundation routes the SMS notification messages through the SMS gateway.

To send SMS notifications from IBM Worklight Foundation, 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 222. `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 223
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 223. `<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.

Table 223. <gateway> element attributes (continued)

Attribute	Attribute Value
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"
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/XMLSch

  <gateway hostname="myhost" id="kannelgw" port="13011" programName="cgi-bin/sendsms" toParamName="t
   <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 and the Command Line Interface for IBM Worklight Developers. The tasks in this section are used to build and deploy your applications, adapters, and projects.

IBM Worklight Foundation provides a set of Ant tasks that help you build and deploy adapters and applications to your 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 Foundation" on page 13.

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}" destinati
    <app-builder applicationFolder="${wl.project.location}/apps/${wl.application.name}" ou
  </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" worklightSer
      <app-deployer deployable="${output.location}/${wl.application.name}-all.wlapp" wor
  </target>
</project>
```

Building applications and adapters

The Ant tasks used for building Worklight applications and adapters are documented in this section.

The following sections document examples of Ant XML files used to build applications and adapters.

Note: Since IBM Worklight Foundation V6.2.0, the `worklight-ant-builder.jar` file is included in the Command Line Interface for IBM Worklight Developers, whereas in earlier versions, it was included in Worklight Server. By default, `worklight-ant-builder.jar` is installed in the following location: *<CLI Install Path>*/public/worklight-ant-builder.jar. For example, on OSX, the default CLI Install Path is /Applications/IBM/Worklight-CLI. If you use the default installation path, the Ant task is installed here: /Applications/IBM/Worklight-CLI/public/worklight-ant-builder.jar.

Building a hybrid application

The Ant task for building a hybrid 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_CLI_INSTALL_DIR/public/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 a native API application

The Ant task for building a native API application has the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_CLI_INSTALL_DIR/public/worklight-ant-builder.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <native-app-builder
      sourcefolder="application-source-files-folder"
      outputFolder="output-folder"/>
  </target>
</project>
```

The `<native-app-builder>` element has the following attributes:

- The `sourceFolder` attribute specifies the root folder for the application, which contains the `application-descriptor.xml` file and other source files for the application.
- The `ouptputFolder` attribute specifies the folder to which the resulting `.wlapp` file is written.

Building an adapter

The Ant task for building an adapter has the following structure:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="WL_CLI_INSTALL_DIR/public/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 that are used for deploying Worklight applications and adapters are documented in this section.

The following sections show examples of Ant XML files that use the `<wladm>` Ant task to deploy applications and 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, you must deploy the corresponding Worklight project of the application before you use this Ant task. For more information, see "Deploying the project WAR file" on page 1105.

A typical Ant script 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/deployers/antlib.xml">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <wladm url="https://server-address:secure-port/worklightadmin" user="username" password="password
      <deploy-app runtime="project-name" file="myApp.wlapp"/>
    </wladm>
  </target>
</project>
```

The `<wladm>` element has the following attributes:
- The `url` attribute (mandatory) specifies the full URL of your Worklight Server administration services web application.
- The `user` attribute (mandatory) and the `password` attribute (mandatory) contain the credentials of a user that is in the role **worklightadmin** or **worklightdeployer**.

The `<deploy-app>` element has the following attributes:
- The `runtime` attribute (mandatory) specifies the name of the Worklight runtime / project.
- The `file` attribute (mandatory) contains the `.wlapp` file to deploy.

For more information about `<wladm>`, see "Administering Worklight applications through ANT" on page 1258.

If you must deploy more than one `.wlapp` file, either add a `<deploy-app>` element for each file in a single `<wladm>` element, or add a `<wladm>` element for each file.

Deploying an adapter

Note: As a prerequisite step, you must deploy the corresponding Worklight project of the adapter before you use this Ant task. For more information, see "Deploying the project WAR file" on page 1105.

A typical Ant script 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/deployers/antlib.xml">
    <classpath>
      <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
    </classpath>
  </taskdef>
  <target name="target-name">
    <wladm url="https://server-address:secure-port/worklightadmin" user="username" password="passw
      <deploy-adapter runtime="project-name" file="myAdapter.adapter"/>
    </wladm>
  </target>
</project>
```

The `<wladm>` element has the following attributes:
- The `url` attribute (mandatory) specifies the full URL of your Worklight Server administration services web application.
- The `user` attribute (mandatory) and the `password` attribute (mandatory) contain the credentials of a user that is in the role **worklightadmin** or **worklightdeployer**.

The `<deploy-adapter>` element has the following attributes:
- The `runtime` attribute (mandatory) specifies the name of the Worklight runtime / project.
- The `file` attribute (mandatory) contains the `.adapter` file to deploy.

For more information about `<wladm>`, see "Administering Worklight applications through ANT" on page 1258.

If you must deploy more than one `.adapter` file, either add a `<deploy-adapter>` element for each file in a single `<wladm>` element, or add a `<wladm>` 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 Foundation 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 Foundation, and described in "Ant tasks for building and deploying applications and adapters" on page 1190 and "Ant tasks for deploying a project WAR file and configuring an application server" on page 1109.

- Use Worklight Console to manually deploy apps and adapters.

You can deploy customer-specific content (apps and adapters) only after the project and Worklight administration WAR files are deployed and the server is started.

If only one project is deployed on the server, you see the Catalog page of Worklight Console and you can start performing administration tasks. If several projects are deployed on the server, you see a list of projects in Worklight Console. Select a project to navigate to the Catalog page of the project.

Note: Only the most recently accessed application is displayed on the Worklight Development Server at run time.
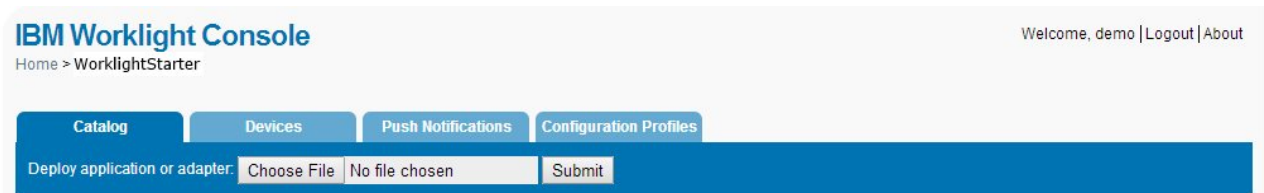
Figure 134. Worklight Console Catalog page

Procedure

1. To deploy an **adapter**, click **Choose File**. Then navigate to the `.adapter` file and select it.
2. Click **Submit**.

   A message is displayed indicating whether the deployment action succeeded or failed.
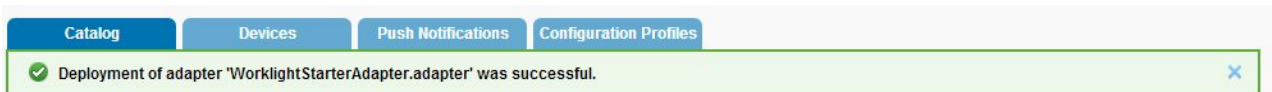
Figure 135. Message for successful adapter deployment

As a result, the details of the deployed adapter are added to the catalog:
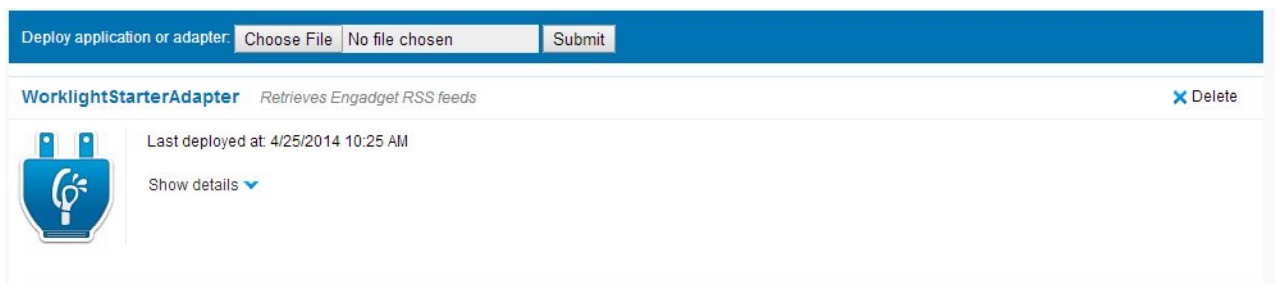
Figure 136. Adapter catalog

3. Click **Show details** to view connectivity details for the adapter and the list of procedures.

Figure 137. Adapter connectivity and procedure details

4. Repeat steps 1 to 3 for each adapter that you want to deploy.

5. To deploy an **application**, in the catalog page, click **Choose file**. Then navigate to the `.wlapp` file and select it.

Figure 138. Catalog page for accessing the list of applications and adapters

6. Click **Submit**. A message is displayed indicating whether the deployment action succeeded or failed.

Figure 139. Message for successful application deployment

As a result, the details of the deployed application are added to the catalog.

Figure 140. Details of the deployed application

7. Repeat steps 5 and 6 for each app that you want to deploy.

Administering adapters and apps in Worklight Console

You administer adapters and apps through Worklight Console.

About this task

Before performing any of the other tasks in this collection of topics, open Worklight Console:

Procedure

1. Open a browser and enter a URL of the following form: https://
   hostname:*secure-port*/worklightconsole where *secure-port* depends on your server
   configuration. The defaults are 9443 for WebSphere Application Server and 8443
   for Apache Tomcat.

Note: Security warning. If you access Worklight Console through http instead of https, your Worklight administration user password will be compromised.

This usage is different from the Worklight Development Server in Worklight Studio, where no security is used. In the development environment, you use the port for the Liberty profile server in the URL: http://localhost:10080/worklightconsole.

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

If only one project is deployed on the server, you see the Catalog page of Worklight Console and you can start performing administration tasks.

If several projects are deployed on the server, you see a list of projects in Worklight Console. Select the project to administer to navigate to the Catalog page of this project.
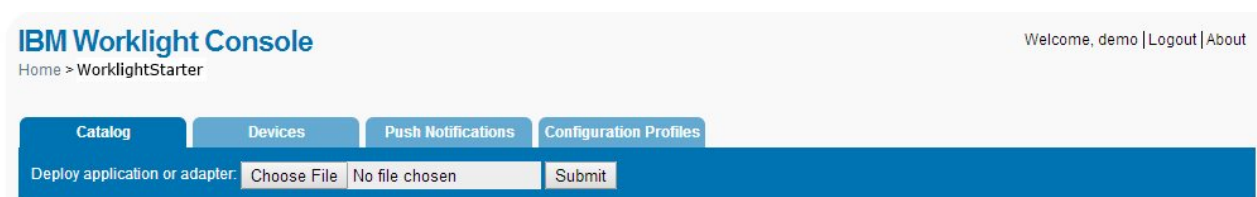
Figure 141. Worklight Console page for choosing an adapter or an app

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**.
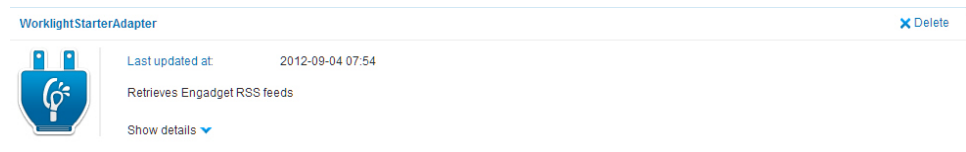
A message is displayed indicating whether the deployment action succeeded or failed. If it succeeded, the details of the deployed adapter are added to the catalog.
3. Click **Show details** to view the connectivity details for the adapter and the list of procedures it exposes.
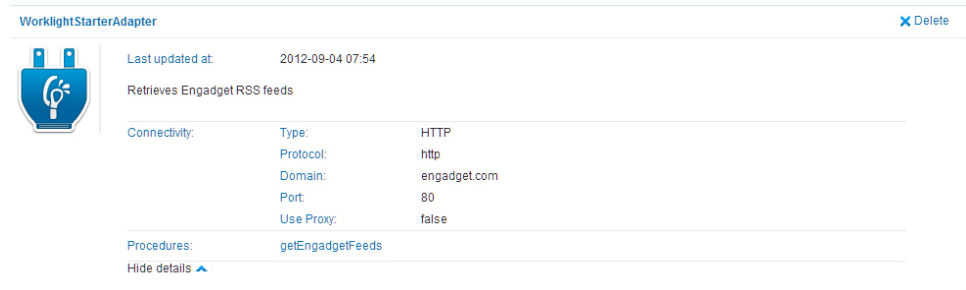
Results

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."

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
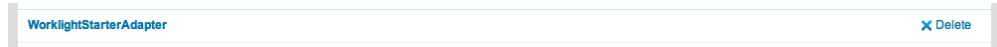
Worklight security overview

Worklight has comprehensive support for various authentication and authorization methods.

Worklight security basics

The following image shows the authentication elements hierarchy:
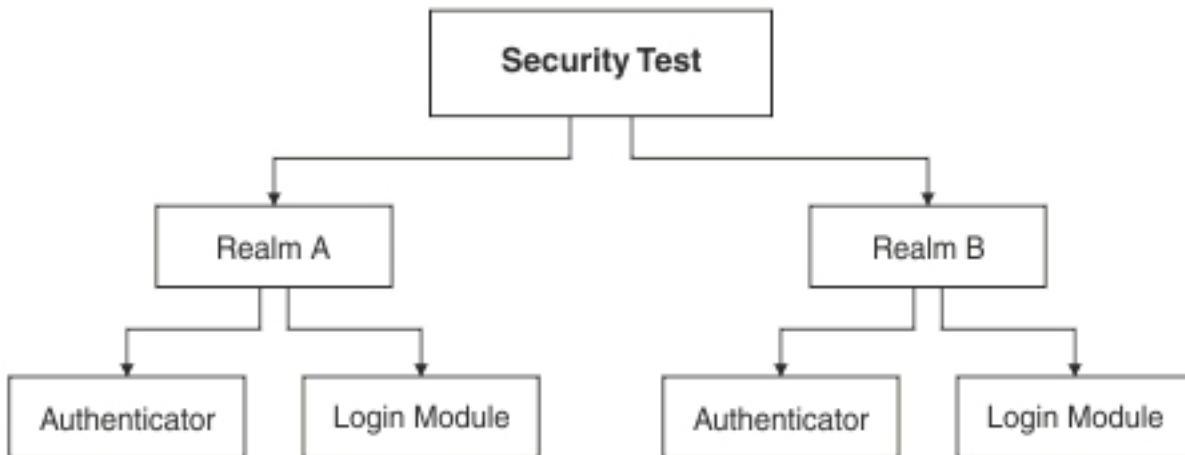
Security test
> A security test is a set of tests that are used to protect a resource, such as an adapter procedure or application environment. A test includes information about which realm is required to authenticate and other parameters, such as authentication order. A protected resource is accessible 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 870.

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 872.

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 874.

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 874.

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>
```

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 in hybrid applications* under category 9, *Authentication
and security*, in Chapter 4, "Tutorials and samples," on page 35.

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:
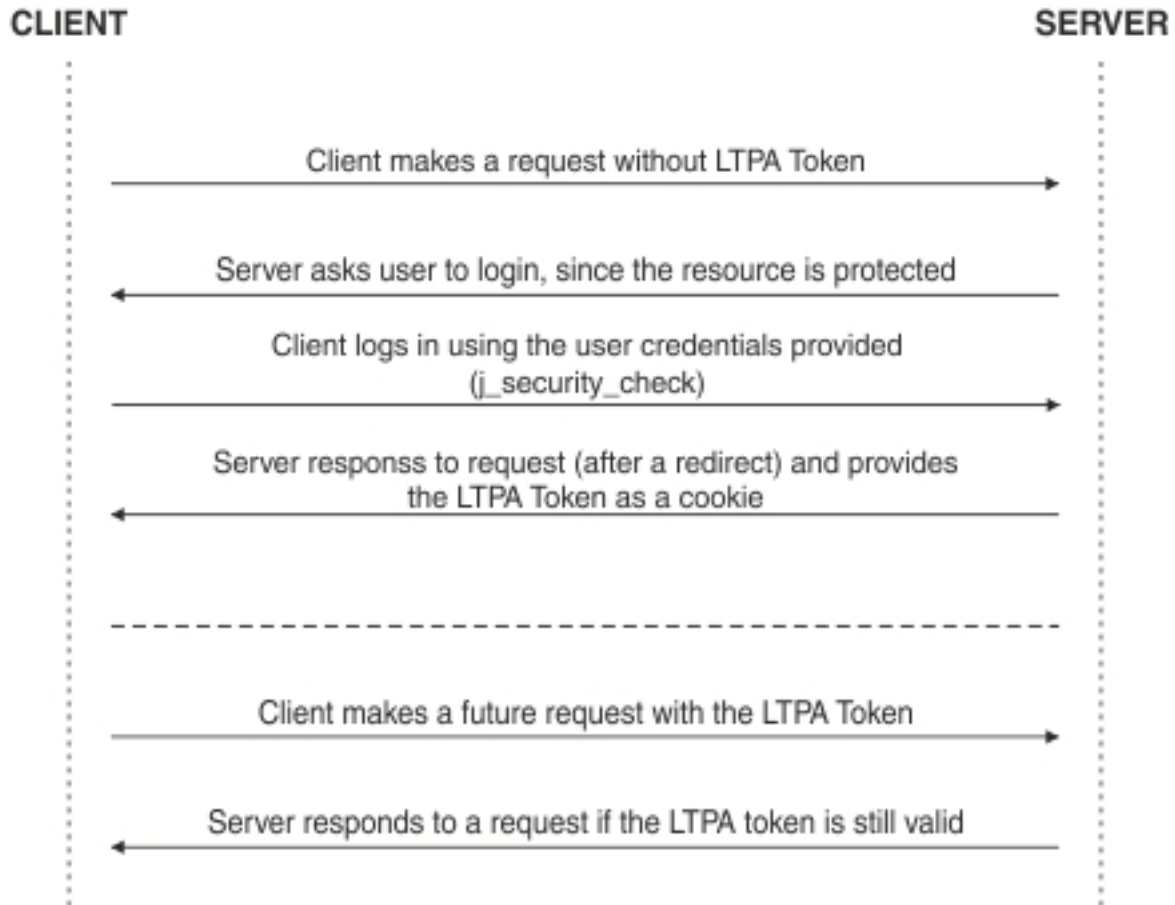
After a user logs in, the server generates an LTPA token, which is an encrypted hash that contains authenticated user information. The token is signed by a private key that is shared among all the servers that want to decode it. The token is usually in cookie form for HTTP services. By sending the token 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:
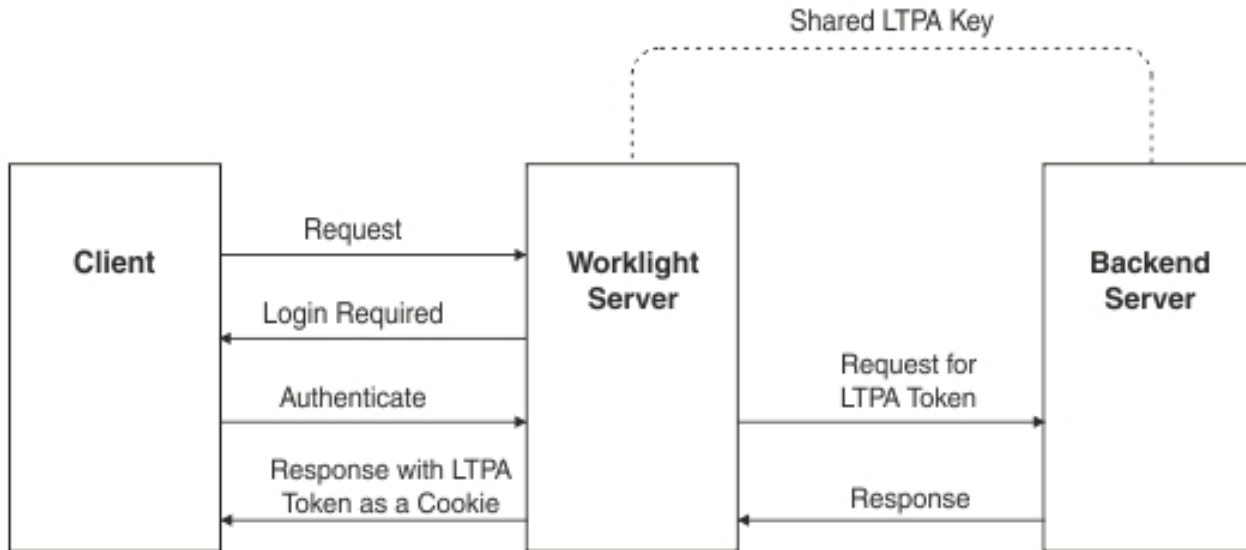
Your infrastructure can also use the LTPA token to communicate with a backend server to act on behalf of the user. The user cannot directly access the backend server. Enterprise environments should use a reverse proxy, such as DataPower or IBM Security Access Manager, in the DMZ, and place the Worklight 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 1215.

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>
```

Configuring the Worklight Server for Trusteer

Configure the Worklight Server to use Trusteer-generated data to protect resources.

About this task

You must update the `authenticationConfig.xml` file to configure your server to use the Worklight Trusteer 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 `trusteerFraudDetectionLogin`:

```
<loginModules>
...
  <loginModule name="trusteerFraudDetectionLogin">
    <className>com.worklight.core.auth.ext.TrusteerLoginModule</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 wl_basicTrusteerFraudDetectionRealm:

```
<realms>
...
  <realm name="basicTrusteerFraudDetectionRealm" loginModule="trusteerFraudDetectionLogin">
    <className>com.worklight.core.auth.ext.TrusteerAuthenticator</className>
      <parameter name="rooted-device" value="block"/>
      <parameter name="device-with-malware" value="block"/>
      <parameter name="rooted-hiders" value="block"/>
      <parameter name="unsecured-wifi" value="alert"/>
      <parameter name="outdated-configuration" value="alert"/>

  </realm>
...
</realms>
```

The possible values for Trusteer realm parameters are described in Table 224.

Table 224. Possible values for Trusteer realm parameters

Value	Description
block	Access fails.
alert	Access is permitted and it is recommended to issue a warning.
accept	Access is permitted.

The error codes that have been defined for Trusteer correspond to the parameters in the realm. See Table 225.

Table 225. Trusteer error codes

Code	Description	Corresponding parameter
TAS_ROOT	Indicates that the device is rooted (Android) or jailbroken (iOS).	rooted-device
TAS_MALWARE	Indicates that the device contains malware. Currently financial malware is detected, but will be expanded to all malware.	device-with-malware
TAS_ROOT_EVIDENCE	Indicate that the device contains root hider applications that hide the fact that the device is rooted/jailbroken.	rooted-hiders
TAS_WIFI	Indicates that the device is currently connected to an unsecured Wi-Fi.	unsecured-wifi

Table 225. Trusteer error codes (continued)

Code	Description	Corresponding parameter
TAS_OUTDATED	Indicates that Trusteer SDK configuration has not updated for some time, meaning that it did not connect to the Trusteer server.	outdated-configuration
TAS_INVALID_HEADER	Indicates that the format of the Trusteer header is invalid.	-
TAS_NO_HEADER	Indicates that the Trusteer SDK is not installed, or has failed to initialize.	-

3. Define a security test in the `<securityTest>` element in the `authenticationConfig.xml` file. For Trusteer, it could be:

```
<customSecurityTest name="TrusteerTest">
  <test realm="wl_basicTrusteerFraudDetectionRealm" isInternalUserID="true" step="1"/>
  ...
</customSecurityTest>
```

4. Use the security test to protect a resource. For example, you can protect an application's environment, such as iPhone, iPad, or Android, completely with that security test by adding the `securityTest` attribute to the environment's element in the `authenticationConfig.xml` file:

```
<android version="1.0" securityTest="TrusteerTest">
  ...
</android>
```

This definition requires every Android device that connects to the server through your application to log in to the `TrusteerTest` security test.

5. Using the same security test, another option is to protect a Worklight adapter procedure. . If you have an adapter procedure named `GetSecretData`, you can protect it in the XML configuration file of the adapter by adding the `<realms>` attribute to the `<procedure>`:

```
<procedure name="GetSecretData" securityTest="TrusteerTest" />
```

6. Create a challenge handler for your Worklight app to handle Trusteer challenges. The following are samples of simple challenge handlers:

JavaScript
```
var trusteerChallengeHandler = WL.Client.createWLChallengeHandler("wl_basicTrusteerFra

trusteerChallengeHandler.handleFailure = function(error) {
  //Note: error object includes array of alerts (same values as error.reason) from the
  //Trusteer authenticator and can be accessed via error.alerts
  WL.SimpleDialog.show("Error", "Operation failed. Please contact customer support (re
  [{text:"OK"}]);
};


//In case authenticator succeeds, there may still be alerts that developer should noti

trusteerChallengeHandler.processSuccess = function(identity){
  var alerts = identity.attributes.alerts;  //Array of alerts codes
  if(alerts.length > 0) {
    WL.SimpleDialog.show("Warning", "Please note that your device is : " + alerts,  [{
  }
}
```

Java

```java
public class TrusteerChallengeHandler extends WLChallengeHandler {

  private static Logger logger = Logger.getInstance(TrusteerChallengeHandler.class.getSim
  public TrusteerChallengeHandler(String realmName) {  super(realmName);    }


    @Override
    public void handleSuccess(JSONObject identity) {
      try {
        JSONArray alerts = identity.getJSONObject("attributes").getJSONArray("alerts");
        if(alerts.length() > 0) {
          logger.warn ("TrusteerChallengeHandler.handleSuccess with alerts: " + alerts);
          //todo: display message to the user
        }
      } catch (Exception e) {
        logger.error("Unexpected error: " + e);
      }
    }


    @Override
    public void handleFailure(JSONObject error) {
      try {
        String errorReason = error.getString("reason");
        logger.error("TrusteerChallengeHandler.handleFailure: " + errorReason + "(" + err
        String msg = "Trusteer fraud detection failed due to " + errorReason;
        JSONArray alerts = error.getJSONArray("alerts");
        if(alerts.length() > 0) {
          logger.warn ("TrusteerChallengeHandler.handleSuccess with alerts: " + alerts);
          //todo: We also have alerts...
        }
        //todo: display error message to user
      } catch (Exception e) {
        logger.warn ("Unexpected error: " + e);
      }
    }

    @Override
    public void handleChallenge(JSONObject challenge) {
      //Nothing to do...
    }
}

// Register your newly created challenge handler for your Trusteer realm:
WLClient.getInstance().registerChallengeHandler(
  new TrusteerChallengeHandler("wl_basicTrusteerFraudDetectionRealm")
);
```

Objective-C

```objc
// Assuming you have added a Trusteer realm to the authentication configuration file of
// your server, you can register a challenge handler to receive the responses from
// the authenticator.

// Create a class that extends WLChallengeHandler:
#import "WLChallengeHandler.h"
@interface TrusteerChallengeHandler : WLChallengeHandler
@end

// Register your newly created challenge handler for your Trusteer realm:
[[WLClient sharedInstance] registerChallengeHandler:
[[TrusteerChallengeHandler alloc] initWithRealm:@"
wl_basicTrusteerFraudDetectionRealm"]];

// If you have set one of your realm options to block, a blocking event will trigger hand
@implementation TrusteerChallengeHandler
//...
```

```
          -(void) handleFailure: (NSDictionary *)failure{
            NSLog(@"Your request could not be completed. Reason code: %@",
              failure[@"reason"]);
          }
          //...
          @end

          // If your have set one of your realm options to alert, you can catch the alert event
          // by implementing the handleSuccess method.
          @implementation TrusteerChallengeHandler
          //...
          -(void) handleSuccess:(NSDictionary *)success{
              NSArray* alerts = success[@"attributes"][@"alerts"];
              if(alerts && alerts.count){
                  for(NSString* alert in alerts){
                      NSLog(@"This device is %@", alert);
                  }
              }
          }
          //...
          @end
```

Accessing Trusteer risk assessment

Access Trusteer risk assessment to add Trusteer protection on the client side.

For an application that is running on a rooted device, you might want to disable the "Transfer Funds" button entirely, in addition to the server-side security tests described in "Configuring the Worklight Server for Trusteer" on page 1207.

The following are code samples for JavaScript, Java, and Objective-C:

JavaScript

```
WL.Trusteer.getRiskAssessment(onSuccess);
```

Where `onSuccess` is a function that receives a JSON object that contains all the data processed by Trusteer. See Trusteer documentation for information on each risk item.

```
function onSuccess(result){
  //See the logs for full result
  WL.Logger.debug(JSON.stringify(result));
  //Check for a specific flag
  if(result["os.rooted"]["value"] != 0){
    alert("This device is rooted!");
  }
}
```

Objective-C

```
#import "WLTrusteer.h"
NSDictionary* risks =[[WLTrusteer sharedInstance] riskAssessment];
```

This returns an `NSDictionary` of all the data that is processed by Truster. See Trusteer documentation for information on each risk item.

```
//See logs for full result
NSLog(@"%@",risks);
//Check for a specific flag
NSNumber* rooted = [[risks objectForKey:@"os.rooted"] objectForKey:@"value"];
if([rooted intValue]!= 0){
 NSLog(@"Device is jailbroken!");
}
```

Java

```
WLTrusteer trusteer = WLTrusteer.getInstance();
JSONObject risks = trusteer.getRiskAssessment();
```

This returns an JSONObject of all the data that is processed by Truster. See Trusteer documentation for information on each risk item.

```
JSONObject rooted = (JSONObject) risks.get("os.rooted");
if(rooted.getInt("value") > 0){
 //device is rooted
}
```

Supported configurations for LTPA

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 4, "Tutorials and samples," on page 35. 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:
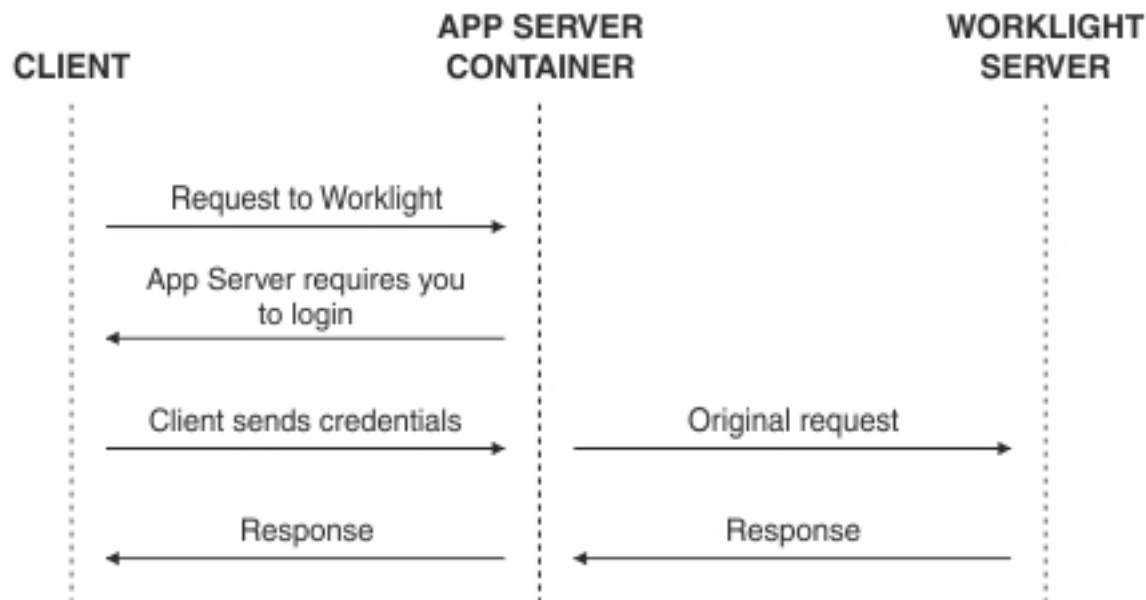
This option is not preferred for new configurations. The application server such as the IBM WebSphere Application Server Liberty profile (Liberty) protects all resources and forces users to log in before any other authentication mechanism. The behavior occurs regardless of the expected authentication 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 1206.

Protective 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:
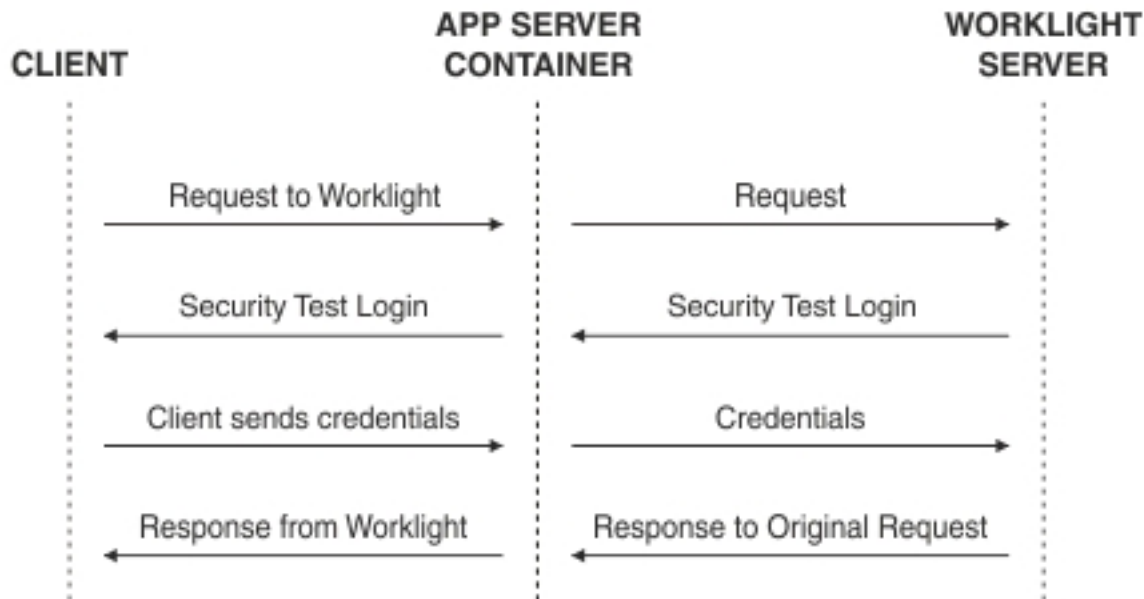
You need to modify only Liberty's `server.xml` file to configure this option. The `WASLTPARealm` handles the actual authentication against the user registry that is defined in the `server.xml` file. The example configuration allows the user with the user name `sample user` and the `password demo to authorize 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 1206.

Advanced security features

IBM Worklight Foundation supports more features that can use LTPA in advanced scenarios, such as user certificate authentication and role-based authentication.

Role-based authentication

In IBM 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 902.

User certificate authentication

In IBM 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 1447.

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 Worklight with a reverse proxy, see "Integration with reverse proxy" on page 1469.

The following image shows a reverse proxy flow:
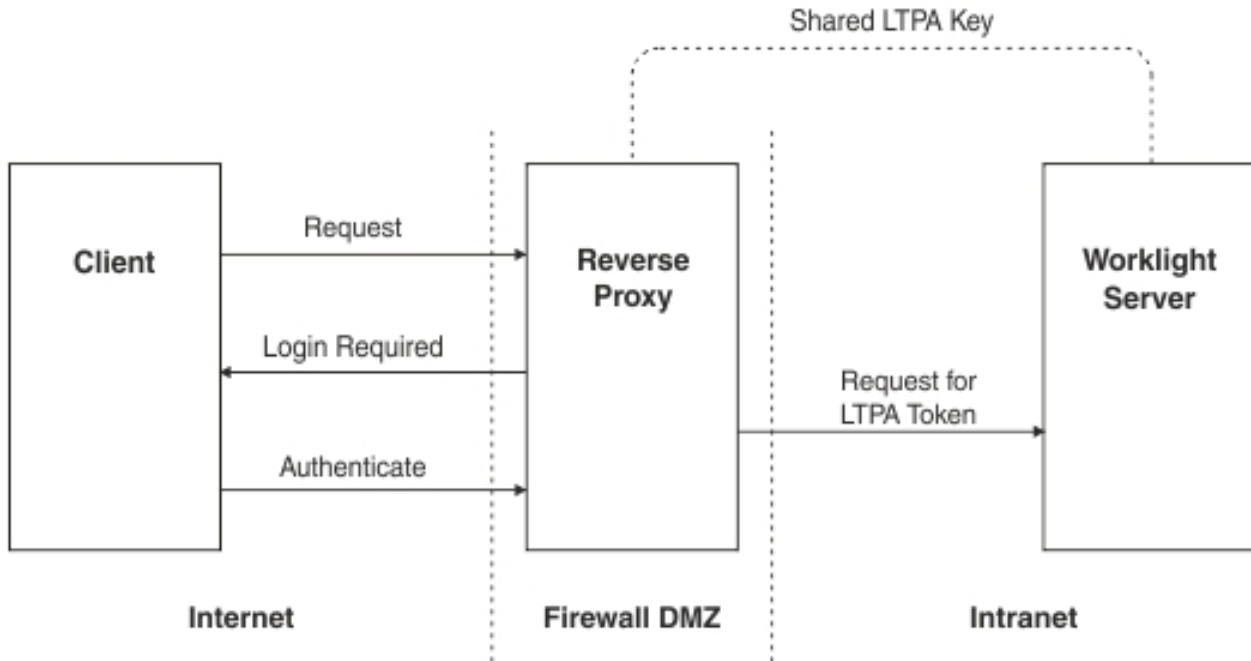
High availability

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 management data is in the Worklight administration services, which verify that all cluster nodes have the same data. With WebSphere Application Server Network Deployment, you can use built in clustering support for distributing the Worklight project WAR (and the Worklight Shared library). For more information, see the IBM WebSphere Application Server V8 user documentation.

Worklight Server can run on a VMware virtual machine. In such cases, one machine image is created and then deployed again and again.

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:

- Mandatory: Sticky session
- Mandatory: 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 Foundation 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 Foundation servers and the IBM Worklight Foundation database.

- 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 Worklight servers, the number of connections might be higher than usual.

Disaster Recovery Site

IBM Worklight Foundation 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 Foundation has such a site for some of its customers.

Within the site, IBM Worklight Foundation provides redundancy at every level: compensating load balancers, multiple IBM Worklight Foundation 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.

Data mirroring
> For the backup site to work, data on the master site must be mirrored to the backup regularly:

Table 226. Data mirroring

Component	Description	Mirror frequency
IBM Worklight Foundation 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 Foundation Software, customization, and content	Any change in IBM Worklight Foundation 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 Worklight apps in production

There are general guidelines for upgrading your Worklight apps when they are already in production, on the Application Center or in app stores.

Deploying your 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 1102. 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 1252 and "Defining administrator messages from Worklight Console in multiple languages" on page 1253.

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 1252 and "Defining administrator messages from Worklight Console in multiple languages" on page 1253.
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 1249 and "Remotely disabling application connectivity" on page 1249.

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 1252 or "Defining administrator messages from Worklight Console in multiple languages" on page 1253.

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 599.

Application authenticity

This feature will not work properly for clients that were built with an older version of Worklight Studio when the application deployed is of a new version of Studio but has the same application version. Those client requests to access the authenticity-protected resources will be denied.

To keep support of old applications using app authenticity, or block them, follow these steps:

1. Upgrade the project using the new Worklight Studio, as described above.

2. Increment the versions of the upgraded applications.

3. Deploy the new WAR file that was built with the new Studio.

4. Deploy the new applications to the server alongside the applications that were built with the old Worklight.

5. Normally, both applications work as expected. If you want to use the new ones only, block the old ones and refer to the new ones for upgrade.

Deploying to the cloud

IBM Worklight Foundation provides the capability to deploy and manage Worklight Server and Worklight applications on IBM PureApplication System, IBM SmartCloud Orchestrator, and IBM PureApplication Service on SoftLayer.

Using IBM Worklight Foundation in combination with IBM PureApplication System, IBM SmartCloud Orchestrator, and IBM PureApplication Service on SoftLayer provides a simple and intuitive environment for developers and administrators to develop mobile applications, test them, and deploy them to the cloud. The following components are available:

IBM Mobile Application Platform Pattern Type
> Provides Worklight runtime and artifacts support for IBM PureApplication System, IBM SmartCloud Orchestrator, and IBM PureApplication Service on SoftLayer.

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.

Compatibility between pattern types and artifacts created with different product versions

If you use Worklight Studio V6.2.0 or earlier to develop your applications, you can upload the associated runtime, application, and adapter artifacts into patterns associated with IBM Worklight Foundation V6.2.0 and later.

Pattern types that are associated with IBM Worklight Foundation V6.2.0 or earlier are not compatible with runtime, application, and adapter artifacts created by using versions of Worklight Studio later than V6.2.0.

For versions V6.0.0 and earlier, only the same versions of server, .war file, application (.wlapp file) and adapters are compatible.

For more information about version compatibility, see "Version compatibility" on page 343.

Installing 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-6.2.0.1.tgz` file in the
`worklight_pattern_6.2.0.1.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 Worklight database workload standards

You need to install custom workload standards for the Worklight runtime database,
administration database, and reports database.

Before you begin

Extract the `WLRTDB.zip`, `WLRPTDB.zip`, and `WLADMDB.zip` files from the
`worklight_pattern_6.2.0.1.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_6.2.0.1.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 do the following:
   a. Upload the `WLRPTDB.zip` file to create the custom reports database workload
      standard.
   b. Upload the `WLADMDB.zip` file to create the administration 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 1225).

Installing new Worklight database workload standard that combines all Worklight tables

In addition to the traditional pattern topology for production environment where you deploy Worklight databases into separate DB2 VMs, Worklight also supports creating a pattern that uses a single DB2 VM. This pattern leverages the new DB Workload that combines all Worklight tables for IBM PureApplication Service on SoftLayer.

Before you begin

Extract the WLALLDB.zip from the worklight_pattern_6.2.0.1.offering.zip file.

Procedure

1. Log in to IBM PureApplication System or IBM PureApplication Service with an account that has permission to create database workload standards.
2. In the Workload Console of PureApplication System or in the home page of PureApplication Service, 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_ALL_DB.
5. From the **Workload type** list, select **Departmental Transactional**.
6. In the **Upload file (.zip)** field, select the WLALLDB.zip file that you retrieved from the worklight_pattern_6.2.0.1.offering.zip file.
   a. Click **Browse**, navigate to the folder into which you extracted the WLALLDB.zip file, and then select the WLALLDB.zip file.
7. Click **Save** to save your Worklight combined database workload standard.

Installing Worklight support for cloud deployment from the command line

If you download the IBM PureApplication System command-line interface, you can install 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 Using the command-line interface in the IBM PureApplication System documentation.

Procedure

Open a command prompt and run the following Python script:

```
./pure -h pure_host_name -u user_name -p password -a  -f install.py worklight_artifacts_path

install.py:
import sys
mobileloc=sys.argv[1]

print "Loading Mobile"
deployer.patterntypes.create(mobileloc+'/worklight.ptype-6.2.0.1.tgz')
```

```
print "Accept/enable"
deployer.patterntypes.get("worklight.ptype","6.2.0.1").acceptLicense()
deployer.patterntypes.get("worklight.ptype","6.2.0.1").enable()

print "creating db workload standards"
temp=deployer.dbworkloads.create({"rate":"3","workload_type":"Departmental OLTP","initial_disk_size"
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")
temp=deployer.dbworkloads.create({"rate":"3","workload_type":"Departmental OLTP","initial_disk_size"
deployer.dbworkloads.get(temp['id']).workloadfiles.upload(mobileloc+"/WLADMDB.zip")
temp=deployer.dbworkloads.create({"rate":"3","workload_type":"Departmental OLTP","initial_disk_size"
deployer.dbworkloads.get(temp['id']).workloadfiles.upload(mobileloc+"/WLALLDB.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 7, "Installing and configuring," on page 57.

Working with the IBM Mobile Application Platform Pattern Type

Working with the IBM Mobile Application Platform Pattern Type involves creating an IBM Mobile Application Platform Pattern, integrating with Tivoli Directory Server, connecting to a Tivoli Directory Server, and managing Worklight VAP instances.

Composition and components

The IBM Mobile Application Platform Pattern Type is composed of the IBM Web Application Pattern and the IBM Mobile Application Platform Pattern. The IBM Mobile Application Platform Pattern provides a number of components.

Composition

IBM Mobile Application Platform Pattern Type is composed of the following patterns:
- IBM Web Application Pattern
- IBM Mobile Application Platform Pattern

Components

In addition to all components provided by IBM Web Application Pattern, IBM Mobile Application Platform Pattern provides the following components:
- Worklight application component
- Worklight adapter component
- Worklight configuration component
- Worklight application component link to enterprise application (WebSphere Application Server) component
- Worklight adapter component link to enterprise application (WebSphere Application Server) component

- Enterprise application (WebSphere Application Server) component link to Worklight configuration component

Importing the sample Worklight virtual application pattern

You import the sample 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_6.2.0.1.offering.zip` called `WorklightSamplePattern.zip`.

Procedure

1. Import the sample Worklight application pattern.
   a. Log on to the IBM PureApplication system console.
   b. Select **Patterns** > **Virtual applications**.
   c. In the filter, select **IBM Mobile Application Platform Pattern Type 6.2**.
   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** and to set the Worklight administration database workload standard for **Worklight Admin 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, an administration 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

Important: 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 a Worklight project to IBM PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer" on page 1234
- "Building a Worklight virtual application" on page 1235

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 227. Worklight Server component properties

Property	Description
Name	Name for the Worklight Server.
EAR file	Worklight EAR file that contains the Worklight Server package to be uploaded.

2. Create Worklight runtime and administration databases, and an optional reports database.

a. From the Assets list, expand **Database Components**, and then drag and drop a Database DB2 component onto the canvas.

b. Supply the following information in the fields provided to define the runtime database:

Table 228. 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 Worklight runtime database (see "Installing custom Worklight database workload standards" on page 1222).

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 229. 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.

Table 229. Worklight reports database component properties (continued)

Property	Description
Source	From the **Source** list, select **Apply a database workload standard**, and then click the database workload standard created for the Worklight reports database (see "Installing custom Worklight database workload standards" on page 1222).

   e. Repeat step 2a to create an administration database.

   f. Supply the following information in the fields provided to define the administration database:

Table 230. Worklight administration database component properties

Property	Description
Name	Name for the Worklight administration database component; for example, WL Admin DB.
Database name	Name for the runtime database; for example, WLADMIN.
Source	From the **Source** list, select **Apply a database workload standard**, and then click the database workload standard created for the Worklight administration database (see "Installing custom Worklight database workload standards" on page 1222).

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 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 Worklight reports database.

   e. Drag a connection from the Worklight Server component to the administration database component.

   f. In the **Resource References of Data Source** field, select **jdbc/WorklightAdminDS** for the Worklight administration 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 231. Worklight configuration component properties

Property	Description
Name	Name for the Worklight configuration component.
Worklight Deployment Username	User name for Worklight application and adapter deployment (the user must have authorization for the deployment).
Worklight Deployment Password	Password for Worklight application and adapter deployment.

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 232. 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 233. 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.

6. Integrate with Tivoli Directory Server or Microsoft Active Directory. For more information, see "Integrating with Tivoli Directory Server" on page 1229.

Creating an IBM Mobile Application Platform Pattern that uses a single DB2 VM

You create an IBM Mobile Application Platform Pattern that uses a single DB2 VM by following the similar procedure of creating a default IBM Mobile Application Platform Pattern. The only difference is that you need to create a combined database rather than creating a runtime database, an administration database, and a reports database.

Before you begin

The following procedure involves creating of a combined database. Ensure that the new Worklight database workload standard is already installed. For more information, see "Installing new Worklight database workload standard that combines all Worklight tables" on page 1223

Procedure

1. Create a Worklight Server. For more information, see "Create a Worklight Server" in "Creating an IBM Mobile Application Platform Pattern" on page 1225.

2. Create Worklight combined database.

   a. From the **Assets** list, expand **Database Components**. Drag a Database DB2 component onto the canvas.

   b. Supply the following information in the fields that are provided to define the combined database:

Table 234. Worklight combined database component properties

Property	Description
Name	Name for the Worklight combined database component. For example, `WL Single DB`.
Database name	Name for the combined database. For example, `WLALL`.
Source	From the **Source** list, select **Apply a database workload standard**, and then click the database workload standard that is created for the Worklight combined database. For more information, see "Installing new Worklight database workload standard that combines all Worklight tables" on page 1223.

3. Configure connectivity for the combined database.

   a. Drag a connection from the Worklight Server component to the combined database component.

   b. In the **Resource References of Data Source** field, select `jdbc/WorklightDS` for the Worklight combined database.

   c. Repeat Step 3(a) to drag another two connections from the Worklight Server component to the combined database.

   d. For the second connection, in the **Resource References of Data Source** field, select `jdbc/WorklightReportsDS` for the Worklight combined database.

   e. For the third connection, in the **Resource References of Data Source** field, select `jdbc/WorklightAdminDS` for the Worklight combined database.

4. Configure the Worklight Server. Create the Worklight applications and the adapters. For more information, see "Configure the Worklight Server" and "Create Worklight applications and adapters" in "Creating an IBM Mobile Application Platform Pattern" on page 1225.

5. Integrate with Tivoli Directory Server or Microsoft Active Directory. For more information, see "Integrating with Tivoli Directory Server."

Integrating with Tivoli Directory Server

Tivoli Directory Server and Microsoft Active Directory (connect out only) are supported as directory servers in IBM Mobile Application Platform Pattern and can be used to protect the Worklight Console in conjunction with **LdapLoginModule** provided by IBM Worklight Foundation.

Since IBM Worklight Foundation V6.2.0, Tivoli Directory Server or Microsoft Active Directory are needed by default to secure the Worklight Console. To use Tivoli Directory Server or Microsoft Active Directory for securing Worklight applications,

LDAPLoginModuleIPAS must be defined in your Worklight application. For more information, see "Integration with Tivoli Directory Server" on page 1235

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, creating four connections from the Worklight Server component to it, and then uploading an LDIF file and setting the Base DN 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 235. 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	Base DN for Tivoli Directory Server to start searching. This value will override the `ldapSearchBase` value in `LDAPLoginModuleIPAS`.
User filter	Specifies the LDAP user filter that searches the existing user registry for users. This value will override the `ldapSearchFilterPattern` value in `LDAPLoginModuleIPAS`.
Group filter	Specifies the LDAP group filter that searches the existing user registry for groups.

4. Create four links from Worklight Server to Tivoli Directory Server for the four Worklight roles (worklightadmin, worklightdeployer, worklightmonitor and worklightoperator). For each link, in **Mapping special subjects**, you can select `AllAuthenticatedUsers`.

Connecting to an existing Tivoli Directory Server:

You connect to an existing Tivoli Directory Server by dragging and dropping an Existing User Registry(IBM Tivoli Directory Server) component onto the PureApplication System canvas, specifying the IP address and port number of your existing Tivoli Directory Server as well as the login DN and password, and creating four connections from the Worklight Server component to the Existing User Registry(IBM Tivoli Directory Server) component.

Procedure
1. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.
2. From the Assets list, drag and drop an Existing User Registry(IBM Tivoli Directory Server) component onto the canvas.

3. Specify the IP address and port number of your existing Tivoli Directory Server, as well as the login DN, Password and Base DN information provided by your server administrator.

4. To build an SSL connection with the external Tivoli Directory Server, select **use SSL** and upload the Server SSL certificate that supports `.cer`, `.crt`, or `.txt` extensions.

5. Create four links from Worklight Server to the Existing User Registry(IBM Tivoli Directory Server) component for the four Worklight roles (worklightadmin, worklightdeployer, worklightmonitor and worklightoperator). For each link, in **Mapping special subjects**, you can select `AllAuthenticatedUsers`.

Connecting to an existing Microsoft Active Directory:

You connect to an existing Microsoft Active Directory by dragging and dropping an Existing User Registry(Microsoft Active Directory) component onto the PureApplication System canvas, specifying the IP address and port number of your existing Microsoft Active Directory, as well as the login DN and password, and creating four connections from the Worklight Server component to the Existing User Registry(Microsoft Active Directory) component.

Procedure

1. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.

2. From the Assets list, drag and drop an Existing User Registry(Microsoft Active Directory) component onto the canvas.

3. Specify the IP address and port number of your existing Microsoft Active Directory, as well as the login DN, Password and Domain suffix of your LDAP information provided by your server administrator.

4. To build an SSL connection with the external Microsoft Active Directory, select **use SSL** and upload the Server SSL certificate that supports `.cer`, `.crt`, or `.txt` extensions.

5. Create four links from Worklight Server to the Existing User Registry(Microsoft Active Directory) component for the four Worklight roles (worklightadmin, worklightdeployer, worklightmonitor and worklightoperator). For each link, in **Mapping special subjects**, you can select `AllAuthenticatedUsers`.

Performing operations on running Worklight Virtual Application Pattern instances

Use the PureApplication System Workload Console to perform management tasks on a running 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 236. Operations on Virtual Application Pattern instances

Operation	Description
Worklight Deployment Username	Username for Worklight application and adapter deployment.
Worklight Deployment Password	Password for Worklight application and adapter deployment.

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".

Integrating by using inbound and outbound connection components

Inbound and outbound connections are supported as optional components in IBM Mobile Application Platform Pattern. You can use inbound and outbound connections, for example, to connect to backend services by opening the firewall ports of the Worklight Server for connection. If your Worklight applications need inbound and outbound connections with backend services, you must add inbound and outbound connection components in the IBM Mobile Application Pattern.

Adding inbound and outbound connection components in the IBM Mobile Application Platform Pattern:

For security reasons, a firewall is enabled on the Worklight Server and only some ports are opened by default for connections with the LDAP server and DB servers. To open additional ports for connection with backend services, you must add inbound and outbound connection components in the IBM Mobile Application Platform Pattern.

Procedure

1. In IBM PureApplication System, in the Virtual Application Builder, click the **Diagram** tab.

2. From the Assets list in the Other Components category, drag and drop a Connect Out (for outbound connection) or a Connect In (for inbound connection) component onto the canvas.

3. In the **Outbound/Inbound Server(s) & Port(s)** field, specify the IP addresses and port numbers for Worklight Server to open its firewall ports for outbound and inbound connections to enterprise backend services. The input must be a comma-separated list of servers and ports; for example, `32543,1.2.3.4:23410`.

4. Create a link from the Worklight Server component to the Connect Out or Connect In component.

Performing operations on inbound and outbound connection components of Worklight Virtual Application Pattern instances:

Use the PureApplication System Virtual Application Console to add or remove outbound or inbound servers and ports for a running 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** on the right side. The Virtual Application Console window opens.

3. In the Virtual Application Console window, click the **Operations** tab, and then from the Operations list, click **FIREWALL**. If there are no outbound or inbound connection component in your IBM Mobile Application Pattern, the **FIREWALL** role is not visible.

4. In the details panel, you can perform the following operations:

Table 237.

Operation	Description
Connect In – Inbound Server(s) & Port(s)	Add or remove servers and ports to be opened for inbound connection in the firewall.
Connect Out – Outbound Server(s) & Port(s)	Add or remove servers and ports to be opened for outbound connection in the firewall.

5. To submit the changes you have made, click **Submit**.

6. In the Operation Execution Results window, verify that the status of your task is displayed as "success". The status is displayed after a few seconds.

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 1224.

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 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 238. 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 a Worklight project to IBM PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer

You deploy a Worklight project to PureApplication System, SmartCloud Orchestrator, or PureApplication Service on SoftLayer 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 1171.

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. In the Edit Configuration window, select the Worklight applications and adapters to be deployed in the Worklight Project Configuration area.
4. In the Database Option area, click **Combined DB** to combine all the Worklight tables into one DB2 VM, or click **Multiple DB** and then decide further whether to include **Report DB** or not. (You can not generate a report afterwards if you decided not to include **Report DB** here), and then click **Run**.

5. 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 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

Since IBM Worklight Foundation V6.2.0, an LDAP server is needed to protect the Worklight Console. To protect also your Worklight applications on PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer, you can either leverage a new User Registry TDS component as a user registry or connect to an existing user registry server.

To integrate with LDAP, you need to implement the LDAP login module in the `authenticationConfig.xml` file located in `WEB-INF\classes\conf` of Worklight runtime WAR 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`. If you connect to a new TDS, its value will be automatically overridden by the built-in TDS server. URL.

  This is an example of a suitable LDAP login module:

```
<loginModule name="LDAPLoginModuleIPAS">
 <className>com.worklight.core.auth.ext.LdapLoginModule</className>
 <parameter name="ldapProviderURL" value="ldaps://192.0.2.123:636"/>
 ...
 ...
</loginModule>
```

- If **Connect to a new TDS** is enabled in your Worklight project configuration, you need to specify a `.ldif` file or use the default `.ldif` file that is used to protect the Worklight Console.
- If **Connect to existing TDS** is enabled, the value of the `ldapProviderURL` parameter is taken as the remote LDAP server address.

Building and deploying Worklight virtual applications by using the command-line interface

IBM Mobile Application Platform Pattern includes a set of Ant tasks to help you build Worklight virtual applications and artifacts and deploy to IBM PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer.

Building a Worklight virtual application

You can use an Ant task to build a Worklight virtual application.

Before you begin

The Ant tasks are contained in the `worklight-ant.jar` file, which you can find in the `worklight_pattern_6.2.0.1.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 239. Ant task build attributes

Attributes	Description
`worklightWar`	Required. The Worklight Console WAR file including the full file path. Note that `worklight-management-ui.war` and `worklight-management-service.war` must be retrieved from Worklight Studio (under {$workspace}/WorklightServerConfig/ servers/worklight/apps) and put in the same folder as the Worklight runtime WAR file.
`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. If not set, the default `ldifFile` is used to connect to Tivoli Directory Service.
`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.

Table 239. Ant task build attributes (continued)

Attributes	Description
`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.
`databaseOption`	Optional. Default value is `Multiple`. In this case, each database runs on a separate DB2 VM. You can set its value to `Combined`, in which case all the databases run in a single DB with all Worklight tables.
`reportDBFlag`	Optional. Whether to include a Report DB if `databaseOption` is set to `Multiple`. For combined DB, it must be `true`. Default value: `true`.

Deploying a Worklight virtual application

You can use an Ant task to deploy a Worklight virtual application.

Before you begin

The Ant tasks are contained in the `worklight-ant.jar` file, which you can find in the `worklight_pattern_6.2.0.1.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 240. Ant task deployment attributes

Attributes	Description
`vapZipFile`	Required. Path to the zip file built by vap-builder.
`ipasHost`	Required. URL of IBM PureApplication System.

Table 240. Ant task deployment attributes (continued)

Attributes	Description
`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 to the cloud

You must configure and connect the operational components of the Application Center to deploy the enterprise application on PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer.

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.
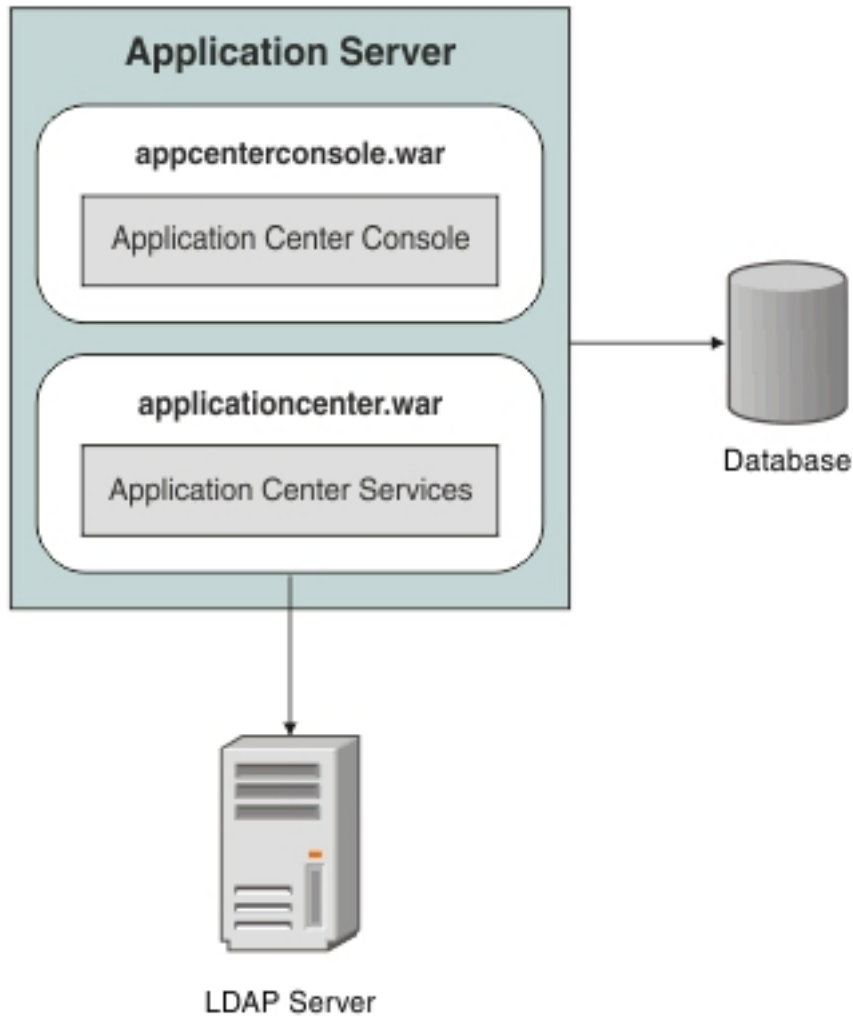
Figure 142. Typical operational model of the Application Center

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

This procedure involves working with the `ApplicationCenterEAR.ear` file and the `create-appcenter-db2.sql` file. You can find both files in the `worklight_pattern_6.2.0.1.offering.zip` file under the `AppCenter` folder.

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

With the full artifacts under the `AppCenter` folder of the `worklight_pattern_6.2.0.1.offering.zip` file. you can install applications on your mobile device with the Application Center mobile client. For more information, see "The mobile client" on page 1350.

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, `ApplicationCenterEAR.ear`, for the Application Center. This file is in the `worklight_pattern_6.2.0.1.offering.zip` file under the `AppCenter` folder. 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 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 the `worklight_pattern_6.2.0.1.offering.zip` file under the `AppCenter` folder.

　　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.

Deployment of analytics to the cloud

The IBM Worklight Analytics Platform is delivered as a WAR file. By using the Web Application Pattern Type, you can deploy it to the cloud. The following sections describe the required steps for successfully deploying the WAR file to IBM

PureApplication System, IBM SmartCloud® Orchestrator, or IBM PureApplication
Service on SoftLayer, and configuring the Worklight Server to add its connection to
the Analytics server.

The procedures in the following topics describe the installation of a single
Analytics node on PureApplication System, IBM SmartCloud Orchestrator, or IBM
PureApplication Service on SoftLayer.

Deploying analytics on IBM PureApplication System

Configure the Web Application Pattern and deploy the analytics server on IBM
PureApplication System or IBM PureApplication Service on SoftLayer.

Before you begin

This procedure involves working with the `analytics.war` file. You can find it in the
`worklight_pattern_6.2.0.1.offering.zip` file in the `Analytics` folder.

You must have an IBM PureApplication System or IBM PureApplication Service on
SoftLayer environment, and the privilege to create Virtual Application Pattern
(VAP) and deploy Virtual Application instances.

About this task

By following this procedure, you prepare the Web Application component for
analytics deployment as a Web Application Pattern on PureApplication System.
You configure it, and then you save the configuration and deploy analytics on
PureApplication System as a web application.

Procedure

1. Get the web application archive (WAR) file, `analytics.war`, for analytics. This
   file is in the `worklight_pattern_6.2.0.1.offering.zip` file in the `Analytics`
   folder.
2. Create the Virtual Application Pattern.
   a. Log in to IBM PureApplication System
   b. Select **Workload Console** > **Patterns** > **Virtual Application Patterns**.
   c. Select **Web Application Pattern Type 2.0**.
   d. Click the plus sign (**+**).
   e. Select the **Blank application** template to start from, and then click **Start
      Building**.
3. Add a Web Application component.
   a. Expand **Application Components**.
   b. Drag the Web Application component onto the canvas on the right.
   c. Select the Web Application component in the canvas and specify the path of
      the analytics WAR file by clicking the **Edit** button.
   d. In the **Context root** field, specify the context root name; for example,
      `analytics`, and make a note of it for later use.
4. Add a routing policy. (Recommended for simplifying the later Worklight Server
   configuration)
   a. Move the mouse over the Web Application component and click the plus
      sign (**+**).
   b. Select **Routing Policy**.

c. In the property pane, click **Routing Policy** and specify the **Virtual Host** name. Make a note of the host name because you will use it later.

5. Optional: Add the JVM policy. If you want to set a larger Java heap size, which can improve performance when the analytics server holds a large number of documents, this can be done by setting the JVM policy.

   a. Select **JVM Policy** in the same way that you selected **Routing Policy**.

   b. In the property pane, specify `JVM heap size: 2048MB - 2048MB`.

   c. In the **Class Loader Order** field, make sure **PARENT LAST** is selected.

6. Save the configuration and deploy analytics on PureApplication System.

   a. Save the virtual application. Give it a name; for example, `Analytics Server`.

   b. Return to Virtual Application Patterns. You should see the pattern that you created in this procedure.

   c. Click **Deploy** to deploy Analytics on PureApplication System.

   d. When the deployment is completed, go to **Virtual Application Instances**. Your pattern deployment is now displayed as being in the "Running" state. You can get the analytics server endpoint link by selecting your deployment instance and on the right side, clicking the **Endpoint** button besides the WAS middleware in the Virtual machine perspective section. The analytics console is displayed in your browser.

Results

When you deploy the pattern, the analytics console is available at:

`http://hostname:port/context-root/console`

For example, if you set the routing policy in step 4 and set the context root to "analytics", the following URL applies:

`http://virtual_host_name/analytics/console`

If you do not set the routing policy in step 4 and set the context root to "analytics", the following URL applies:

`http://WAS_IP:9080/analytics/console`

Note:

With the routing policy set, if DNS is not configured with the IP addresses of ELBInstances, the endpoint link does not work properly. You can set up the hosts file on the computer that you are using to access the deployment dashboard to correct the link. The hosts file is typically found in one of the following locations:

UNIX `/etc/hosts`

Windows
> `%SystemRoot%\system32\drivers\etc\hosts`

Obtain the IP address of any ELBInstance and in the same cloud group as the virtual application instance, add an entry to the hosts file. See the following example, where *elbinst_ip* is the ELBInstance IP address.

`virtual_host_name: <varname>elbinst_ip</varname>`

`elbinst_ip: virtual_host_name`

If you deploy any virtual application without a manual routing policy before ELB is deployed and running, the application does not require and use the ELB proxy

shared service. You can access the application from the WAS ENDPOINT; for example `http://WAS_IP:9080/contextRoot`. The WAS ENDPOINT is still accessible even if a proxy is in use. For more information about ELB, see Elastic load balancing proxy shared service.

Configuring Worklight Server and deploying IBM Mobile Application Platform Pattern with an analytics connection

Configure the Worklight Server to add its connection to the Analytics server and deploy it on PureApplication System or IBM PureApplication Service on SoftLayer.

Before you begin

This procedure involves working with the Worklight Server configuration. For evaluation, you can configure it in the `WorklightStarter.ear` file in the `worklight_pattern_6.2.0.1.offering.zip` file, contained in the sample pattern `WorklightSamplePattern.zip` file. For production, you can use the IBM Mobile Application Platform Pattern Extension or the command line interface to package up the Worklight Server into an EAR file and deploy it to the cloud.

You must have an IBM PureApplication System or IBM PureApplication Service on SoftLayer environment, and the privilege to create Virtual Application Pattern (VAP) and deploy Virtual Application instances.

About this task

By following this procedure, you configure the Worklight Server and then deploy it as IBM Mobile Application Platform Pattern with an analytics connection.

Procedure

1. Build your Worklight project in Worklight Studio. Configure the `worklight.properties` file under the `server/conf` path. Set the `wl.analytics.url` property to point to the deployed analytics WAR file.

   `wl.analytics.url=http://<hostname>:<port>/<context-root>/data`

   a. (Recommended) If a manual routing policy for analytics deployment (see "Deploying analytics on IBM PureApplication System" on page 1242) will set the virtual host name to `ibm.com` and the context root is "analytics", the `wl.analytics.url` property is as follows:

      `wl.analytics.url=http://ibm.com/analytics/data`

      In case "a", you can define the Analytics URL in advance and do not need to wait for Analytics deployment completes first.

   b. If you deploy analytics without a manual routing policy before ELB is deployed and running, the application does not require or use the ELB proxy shared service. The `wl.analytics.url` property is as follows:

      `wl.analytics.url=http://<WAS_ip>:9080/analytics/data`

   c. If you deploy analytics without a manual routing policy after ELB is deployed and running, the `wl.analytics.url` property is as follows:

      `wl.analytics.url=http://defaultHost/<random_prefix_context_root>/analytics/data`

      In cases "b" and "c", since `WAS_ip` and `random_prefix_context_root` can only be retrieved after analytics deployment is completed, the Worklight Server must be configured after analytics deployment.

2. Create your IBM Mobile Application Platform Pattern by following the procedure in Creating an IBM Mobile Application Platform Pattern and deploy it to IBM PureApplication System or IBM PureApplication Service on SoftLayer environment. You can also deploy the Worklight project via IBM Mobile

Application Platform Pattern Extension or the command line interface. For more information, see the following topics:

- Deploying a Worklight project to IBM PureApplication System, IBM SmartCloud Orchestrator, or IBM PureApplication Service on SoftLayer
- Building a Worklight virtual application

3. When the deployment is done, go to Virtual Application Instances and select your Worklight deployment instance. In the right side, click the endpoint button of WAS middleware and open the Worklight Console by clicking the endpoint link with context root `worklightconsole`. You can see the Analytics Dashboard link in the right side of Worklight Console.

4. Click the **Analytics Dashboard** link. The analytics console is displayed in your browser. For more information about analytics, see Analytics.

Chapter 13. Administering Worklight applications

Run and maintain Worklight applications in production.

IBM Worklight Foundation provides several ways to administer Worklight applications in development or in production. Worklight Console is the main tool with which you can monitor all deployed Worklight applications from a centralized web-based console.

The main operations that you can perform through Worklight Console are:
- Deploy mobile applications and adapters to Worklight Server.
- Manage application versions to deploy new versions or remotely disable old versions.
- Manage mobile devices and users to manage access to a specific device or access for a specific user to an application.
- Display notification messages on application startup.
- Monitor push notification services.
- Collect client-side logs for specific applications installed on a specific device.

Not every kind of administration user can perform every administration operation. Worklight Console, and all administration tools, have four different roles defined for administration of Worklight applications. The following Worklight administration roles are defined:

Monitor
> In this role, a user can monitor deployed Worklight projects and deployed artifacts. This role is read-only.

Operator
> An Operator can perform all mobile application management operations, but cannot add or remove application versions or adapters.

Deployer
> In this role, a user can perform the same operations as the Operator, but can also deploy applications and adapters.

Administrator
> In this role, a user can perform all application administration operations.

Note: In Worklight Studio V6.2.0, the predefined Worklight Console that is deployed to the embedded Liberty server has the following authentication configuration:
- Role `"worklightadmin"`, user `"admin"`, password `"admin"`
- Role `"worklightdeployer"`, user `"deployer"`, password: `"demo"`
- Role `"worklightmonitor"`, user `"monitor"`, password: `"demo"`
- Role `"worklightoperator"`, user `"operator"`, password: `"demo"`

You must map the different administrators to these roles when you configure Worklight Server during installation of Worklight Console. See "Installing and configuring Worklight Server administration" on page 104.

Worklight Console can be used to administer several runtime environments, which are issued from several independent Worklight projects in Worklight Studio, and that are deployed to the same application server or cluster. For more information about deploying a Worklight project, see "Deploying an application from development to a test or production environment" on page 1102.

Worklight Console is not the only way to administer Worklight applications. IBM Worklight Foundation also provides other tools to incorporate administration operations into your build and deployment process.

A set of REST services is available to perform administration operations. For API reference documentation of these services, see "REST Services API" on page 1016.

With this set of REST services, you can perform the same operations that you can do in Worklight Console. You can manage applications, adapters, and, for example, upload a new version of an application or disable an old version.

Worklight applications can also be administered by using Ant tasks or with the `wladmin` command-line tool. See "Administering Worklight applications through ANT" on page 1258 or "Administering Worklight applications through the command line" on page 1275.

Similar to the web-based console, the REST services, Ant tasks, and command-line tools are secured and require you to provide your administrator credentials, which enable you to perform operations within your specified role.

Administering Worklight applications with Worklight Console

You can administer Worklight applications through the Worklight Console by deploying new versions of mobile and desktop apps, by locking apps or denying access, or by displaying notification messages.

In previous releases of IBM Worklight, Worklight Console was deployed in the project WAR file. If two project WAR files were deployed, each one had its own administration console.

Since IBM Worklight Foundation V6.2.0, Worklight Console is deployed separately and can administer all the runtime environments in the same server.

When the console is launched by entering a URL of the form https://*hostname:secure-port*/worklightconsole in secure mode for production or test, or http://*server-name:port*/worklightconsole in development, the list of all the runtime environments is displayed. Select the runtime environment that you want to administer to access the functions of the Worklight Console.

You can return to the list of runtime environments by clicking the **Home** link in the series of navigational links at the top of the page in Worklight Console.

Use 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. For an easy way to upgrade an application, see "Upgrading a mobile application in Worklight Server and the Application Center" on page 1326. 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.

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 Android, iPhone, iPad, and Windows Phone.

Procedure

To lock an application version for a specific environment, select **Lock this version** for the application version in the relevant environment.
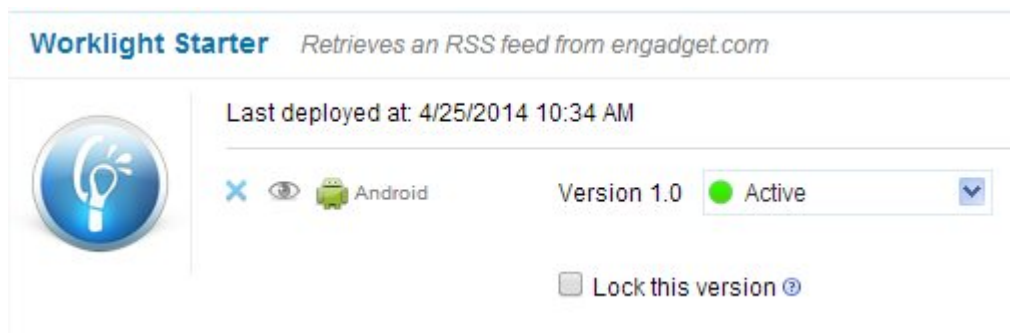
Figure 143. Locking an application version

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 Server 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 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:
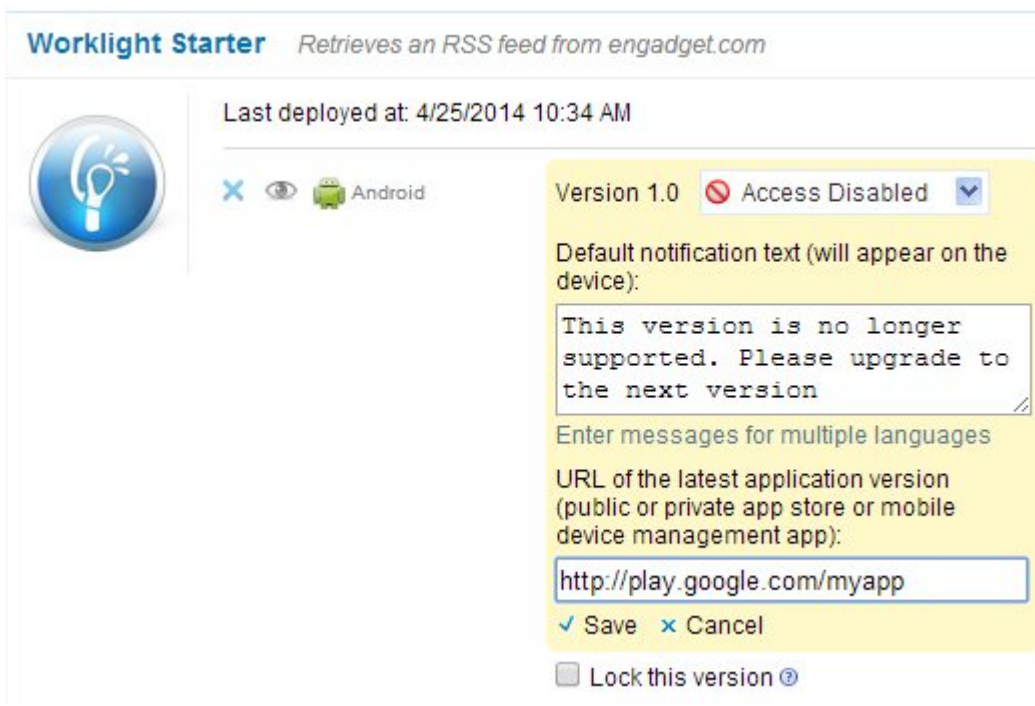
Figure 144. Denying access to older application versions

You can also specify a URL for the new version of the application (usually in the appropriate public or private app store). For some environments, the Application Center provides a URL to access the Details view of an application version directly. See "Application properties" on page 1322.

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.

Figure 145. Denying access to older application versions – message received by user
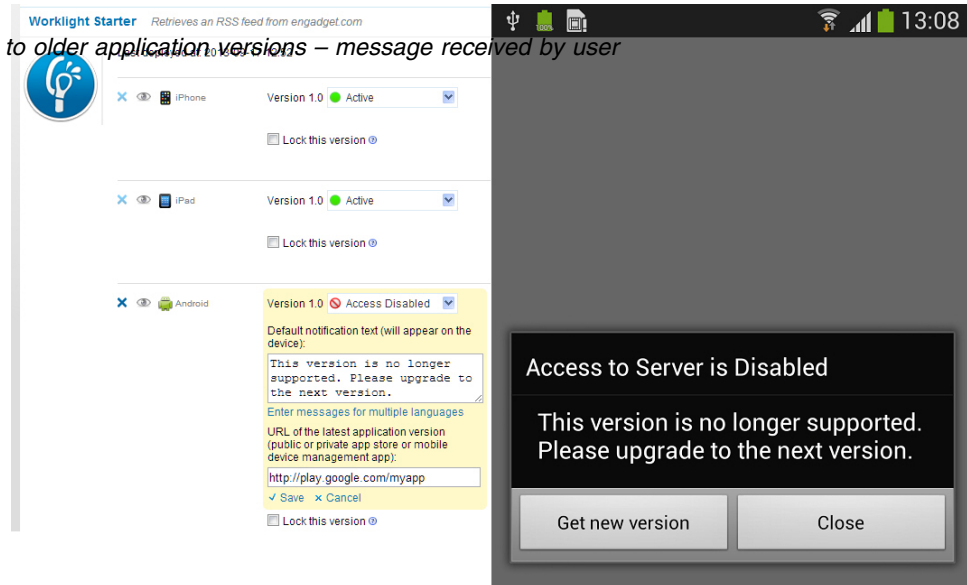
Modifying the behavior of the Remote Disable operation

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 Foundation, 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:

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

1. For the relevant application, change the status of the application version from **Active** to **Active, Notifying**.

2. Add a custom message:
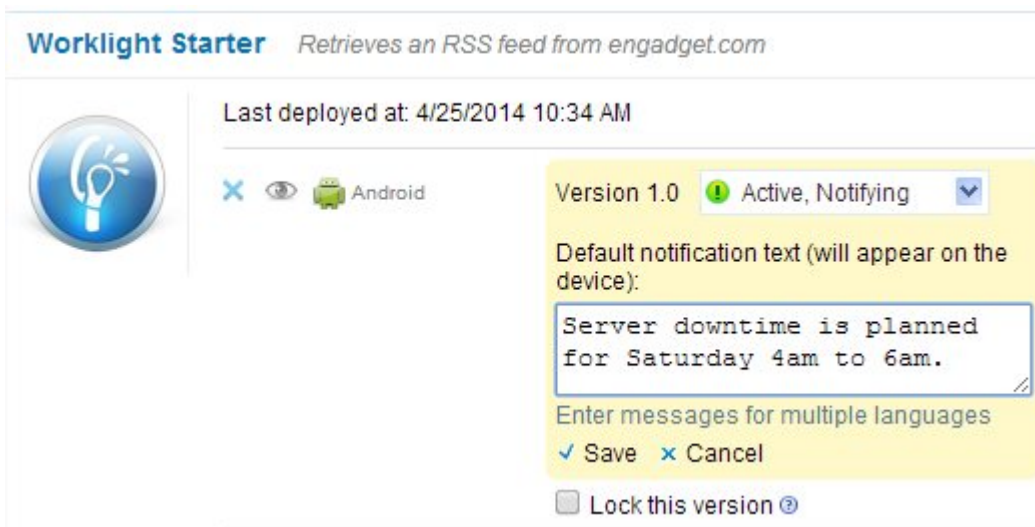
Figure 146. Displaying a notification message

Results

The message is displayed the next time that the app is started or resumed. The message is displayed only once.

Related tasks:

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 **Access Disabled** in the list of application rules.
2. Click **Enter messages for multiple languages**.

Figure 147. Defining the status of application rules in Worklight Console

3. In the **Messages for multiple languages** window that opens, notice that you can upload a CSV file.

Figure 148. 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:

Figure 149. Sample CSV 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.

6. Optional: Click **Clear** to clear the **Supported Languages** list. This action does not clear the default message.
7. Click **Save** to save the messages that you uploaded, or **Cancel** to discard the changes and return to the console.

   Note: If you modified the default message, then the new default message shows.

This figure displays the mobile device of the user, which shows the localized message. The title and the button caption are in English. If the locale does not supply any messages, the default message is returned.
Figure 151. 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 "Worklight security framework" on page 860 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.
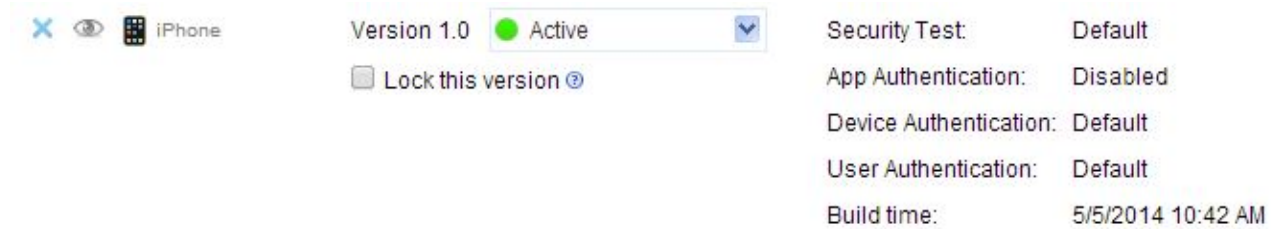
Figure 152. Authenticity testing disabled for iPhone

- If the app is configured with authenticity testing enabled for a specific version, then the Authenticity Testing drop-down menu in the Console is enabled. An example for the Android environment is shown in the following figure.

Figure 153. 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, blocking** – 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 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.

Activity log of operations on runtime environments

Use the activity log to access the status of management operations carried out on the current runtime environment and their effect on the servers.

The activity log displays a list of management operations, initiated from the console or the command line, on the current runtime environment. The most recent operation is displayed first.

You access the activity log by clicking **Activity log** at the bottom of each page in Worklight Console.

If a management operation fails, expand the row that lists the failed operation to access more information about the current state of each server. In this view, only the main error message is shown. To access the complete log, download the activity log by clicking the **Download complete activity log** link.

Activity Log

This page displays management operations on the runtime environment. For more details on errors, download the complete activity log.

Download complete activity log

Refresh

Status	Date	Type	Name	Details	Values
− FAILURE	4/28/2014 3:23 PM	Adapter upload	MyAdapter		

Node	Status	Main error description
server	FAILURE	Preparation to deploy adapter '' threw exception: Procedure 'getStories' in adapter 'MyAdapter' requires security test 'nonExsitence1' which is not defined in authenticationConfig.xml.

Status	Date	Type	Name	Details	Values
SUCCESS	4/28/2014 3:21 PM	Application upload	worklightStarter	Android (1.0), BlackBerry 6 and 7 (1.0), iPhone (1.0), iP...	
SUCCESS	4/28/2014 3:20 PM	Application deletion	HelloWorld		
SUCCESS	4/28/2014 3:20 PM	Application deletion	CommonUIControlsApp		
SUCCESS	4/28/2014 2:27 PM	Application upload	HelloWorld	Android (1.0)	
SUCCESS	4/28/2014 2:26 PM	Application upload	CommonUIControlsApp	Android (1.0)	

Figure 154. Sample activity log

Administering Worklight applications through ANT

You can administer Worklight applications through the `<wladm>` Ant task.

Comparison with other facilities

You can execute administration operations with IBM Worklight Foundation through:
- The Worklight Console, which is interactive.
- The `<wladm>` Ant task.
- The `wladm` program.
- The Worklight administration REST services.

The last three are useful for automated or unattended execution of operations, for example:
- to eliminate operator errors for repetitive operations
- to execute the operations outside the operator's normal working hours

The `<wladm>` Ant task and the `wladm` program are simpler to use and have better error reporting than the REST services. The advantage of the `<wladm>` Ant task over the `wladm` program is that it is platform independent and easier to integrate when an integration with Ant is already available.

Prerequisites

Apache Ant is required to run the `<wladm>` task. For information about the minimum supported version of Ant, see "System requirements for using IBM Worklight Foundation" on page 13.

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.

If you want to use the `<wladm>` Ant task on a machine different from the one on which you installed Worklight Server, you can do so if you:
- Copy the file *WL_INSTALL_DIR*/WorklightServer/worklight-ant-deployer.jar to the machine.
- Make sure a supported version of Apache Ant and a Java runtime environment are installed on the machine.

In order to use the `<wladm>` Ant task, add this initialization command to the Ant script:

```
<taskdef resource="com/worklight/ant/deployers/antlib.xml">
  <classpath>
    <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

Other initialization commands that refer to the same `worklight-ant-deployer.jar` file are redundant because the initialization done by `defaults.properties` are also implicitly done by `antlib.xml`. Here is one example of a redundant initialization command:

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="WL_INSTALL_DIR/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

Invoking the wladm Ant task

You can use the `<wladm>` Ant task and its associated commands to administer Worklight applications.

Invocation

The `<wladm>` Ant task is invoked as follows:

```
<wladm url=... user=... password=...|passwordfile=... [secure=...]>
    some commands
</wladm>
```

Attributes

The `<wladm>` Ant task has the following attributes:

Table 241. List of <wladm> attributes

Attribute	Description	Required	Default
url	Base URL of the Worklight admin services web application	Yes	
secure	Whether to avoid operations with security risks	No	true
user	User name for accessing the Worklight admin services	Yes	
password	Password for the user	Either one is required	
passwordfile	File containing the password for the user		

The URL is the base URL of the Worklight admin services web application, that preferably uses the HTTPS protocol. For example, if you use default ports and context roots, this URL is:

* https://server:9443/worklightadmin for a WebSphere server
* https://server:8443/worklightadmin for a Tomcat server

The secure attribute: `secure="true"` is the default. Setting `secure="false"` means that:

* the user and password might be transmitted in an insecure way (possibly even through unencrypted HTTP),
* the server's SSL certificates are accepted even if self-signed or if they were created for a different hostname than the specified server's hostname.

Specify the password either in the Ant script, through the `password` attribute, or in a separate file, indicated through the `passwordfile` attribute. The password is sensitive information and therefore needs to be protected. You must prevent other users on the same computer from learning this password. To secure the password, before you enter the password into a file, you must remove the read permissions of the file for users other than yourself. For example, you can use one of the following commands:

* On UNIX: `chmod 600` *adminpassword.txt*
* On Windows: `cacls` *adminpassword.txt* `/P Administrators:F %USERDOMAIN%\%USERNAME%:F`

Additionally, you might want to obfuscate the password to hide it from an occasional glimpse. To do so, use the `wladm config password` command to store the obfuscated password in a configuration file. Then, you can copy and paste the obfuscated password to the Ant script or to the password file that you want to use.

The `<wladm>` invocation contains commands that are encoded in inner elements. They are executed in the order in which they are listed. If one of the commands fails, the remaining commands are not executed, and the `<wladm>` invocation fails.

Elements

The following elements can be used in `<wladm>`:

Table 242. Elements that can be used in <wladm>

Element	Description	Count
show-info	Shows user and configuration information	0..∞
show-versions	Shows versions information	0..∞
list-runtimes	Lists the runtimes	0..∞
show-runtime	Shows information about a runtime	0..∞
delete-runtime	Deletes a runtime	0..∞
list-adapters	Lists the adapters	0..∞
deploy-adapter	Deploys an adapter	0..∞
show-adapter	Shows information about an adapter	0..∞
delete-adapter	Deletes an adapter	0..∞
adapter	Other operations on an adapter	0..∞
list-apps	Lists the apps	0..∞
deploy-app	Deploys an app	0..∞
show-app	Shows information about an app	0..∞
delete-app	Deletes an app	0..∞
delete-app-version	Delete a version of an app	0..∞
app-version	Other operations on an app	0..∞
list-devices	Lists the devices	0..∞
remove-device	Removes a device	0..∞
device	Other operations for a device	0..∞

XML Format

The output of most commands is in XML, and the input of specific commands, such as `<set-accessrule>`, is in XML as well. You can find the XML schemas of these XML formats in the directory *WL_INSTALL_DIR*/WorklightServer/wladm-schemas/. The commands that receive an XML response from the server verify that this response conforms to the specific schema. You can disable this check by specifying the attribute `xmlvalidation="none"`.

Output character set

Normal output that is produced by the `<wladm>` Ant task is encoded in the encoding of the current locale. On Windows, this encoding is the so-called "ANSI code page". The effects are as follows:

• Characters outside of this character set are converted to question marks when they are output.

- When a piece of XML is being output, it is still labeled as `<?xml version="1.0" encoding="UTF-8"?>` although in reality the encoding of the output is the encoding of the current locale.
- When the output goes to a Windows command prompt (`cmd.exe`) window, non-ASCII characters are incorrectly displayed because such windows assume characters to be encoded in the so-called "OEM code page".

To work around this limitation:

- On operating systems other than Windows, use a locale whose encoding is UTF-8. This is the default locale on Red Hat Linux and OS X; many other operating systems have a locale that is named `en_US.UTF-8`.
- Or use the attribute `output="some file name"` to redirect the output of a `wladm` command to a file.

Commands for adapters

When you invoke the `<wladm>` Ant task, you can include various commands for adapters.

The <list-adapters> command

The `<list-adapters>` command returns a list of the adapters deployed for a given runtime. It has the following attributes:

Table 243. <list-adapters> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<list-adapters runtime="worklight"/>
```

This command is based on the "Adapters (GET)" on page 1026 REST service.

The <deploy-adapter> command

The `<deploy-adapter>` command deploys an adapter in a runtime. It has the following attributes:

Table 244. <deploy-adapter> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
file	Binary adapter file (.adapter)	Yes	

Example:

```
<deploy-adapter runtime="worklight" file="MyAdapter.adapter"/>
```

This command is based on the "Adapter (POST)" on page 1022 REST service.

The <show-adapter> command

The <show-adapter> command shows details about an adapter. It has the following attributes:

Table 245. <show-adapter> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an adapter	Yes	
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<show-adapter runtime="worklight" name="MyAdapter"/>
```

This command is based on the "Adapter (GET)" on page 1020 REST service.

The <delete-adapter> command

The <delete-adapter> command removes (undeploys) an adapter from a runtime. It has the following attributes:

Table 246. <delete-adapter> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an adapter	Yes	

Example:

```
<delete-adapter runtime="worklight" name="MyAdapter"/>
```

This command is based on the "Adapter (DELETE)" on page 1017 REST service.

The <adapter> command group

The <adapter> command group has the following attributes:

Table 247. <adapter> command group attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an adapter	Yes	

It supports the following elements:

Table 248. <adapter> command group elements

Element	Description	Count
get-binary	Gets the binary data	0..∞

The <get-binary> command

The <get-binary> command inside an <adapter> element returns the binary adapter file. It has the following attributes:

Table 249. <get-binary> command attributes

Attribute	Description	Required	Default
tofile	Name of the output file.	Yes	

Example:

```
<adapter runtime="worklight" name="MyAdapter">
  <get-binary tofile="/tmp/MyAdapter.adapter"/>
</adapter>
```

This command is based on the "Adapter Binary (GET, HEAD)" on page 1016 REST service.

Commands for apps

When you invoke the <wladm> Ant task, you can include various commands for apps.

The <list-apps> command

The <list-apps> command returns a list of the apps that are deployed in a runtime. It has the following attributes:

Table 250. <list-apps> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<list-apps runtime="worklight"/>
```

This command is based on the "Applications (GET)" on page 1054 REST service.

The <deploy-app> command

The <deploy-app> command deploys an app (possibly with multiple environments) in a runtime. It has the following attributes:

Table 251. <deploy-app> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
file	Binary app file (.wlapp, not .apk or .ipa)	Yes	

Example:

```
<deploy-app runtime="worklight" file="MyApp-all.wlapp"/>
```

This command is based on the "Application (POST)" on page 1050 REST service.

The <show-app> command

The <show-app> command returns a list of the apps that are deployed in a runtime. It has the following attributes:

Table 252. <show-app> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an app	Yes	
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<show-app runtime="worklight" name="MyApp"/>
```

This command is based on the "Application (GET)" on page 1046 REST service.

The <delete-app> command

The <delete-app> command removes (undeploys) an app, with all its app versions, for all environments for which it was deployed, from a runtime. It has the following attributes:

Table 253. <delete-app> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an app	Yes	

Example:

```
<delete-app runtime="worklight" name="MyApp"/>
```

This command is based on the "Application (DELETE)" on page 1043 REST service.

The <delete-app-version> command

The <delete-app-version> command removes (undeploys) an app version from a runtime. It has the following attributes:

Table 254. <delete-app-version> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an app	Yes	
environment	Mobile platform	Yes	
version	Version of the app	Yes	

Example:

```
<delete-app-version runtime="worklight" name="MyApp" environment="android"
version="1.1"/>
```

This command is based on the "App Version (DELETE)" on page 1037 REST service.

The <app-version> command group

The <app-version> command group has the following attributes:

Table 255. <app-version> command group attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
name	Name of an app	Yes	
environment	Mobile platform	Yes	
version	Version of the app	Yes	

It supports the following elements:

Table 256. <app-version> command group elements

Element	Description	Count
get-binary	Gets the binary data	0..∞
get-accessrule	Gets the access rule	0..∞
set-accessrule	Changes the access rule	0..∞
get-authenticitycheckrule	Gets the authenticity check rule	0..∞
set-authenticitycheckrule	Changes the authenticity check rule	0..∞
get-lock	Gets the lock state	0..∞
set-lock	Changes the lock state	0..∞

The <get-binary> command

The `<get-binary>` command, inside an `<app-version>` element, returns the binary wlapp file for a version of an app. It has the following attributes:

Table 257. <get-binary> command attributes

Attribute	Description	Required	Default
tofile	Name of the output file.	Yes	

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <get-binary tofile="/tmp/MyApp.wlapp"/>
</app-version>
```

This command is based on the "Application Binary (GET, HEAD)" on page 1042 REST service.

The <get-accessrule> command

The `<get-accessrule>` command returns the access rule for an app version. It has the following attributes:

Table 258. <get-accessrule> command attributes

Attribute	Description	Required	Default
output	Name of a file in which to store the output.	No	
outputproperty	Name of an Ant property in which to store the output.	No	

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <get-accessrule output="/tmp/MyApp-accessrule.xml"/>
</app-version>
```

This command is based on the "Application (GET)" on page 1046 REST service.

The <set-accessrule> command

The `<set-accessrule>` command changes the access rule for an app version. It has the following attributes:

Table 259. <set-accessrule> command attributes

Attribute	Description	Required	Default
file	Name of the input file.	Yes	

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <set-accessrule file="/tmp/new-accessrule.xml"/>
</app-version>
```

This command is based on the "App Version Access Rule (PUT)" on page 1028 REST service.

The <get-authenticitycheckrule> command

The <get-authenticitycheckrule> command returns the authenticity check rule for an app version. It has the following attributes:

Table 260. <get-authenticitycheckrule> command attributes

Attribute	Description	Required	Default
output	Name of a file in which to store the output.	No	
outputproperty	Name of an Ant property in which to store the output.	No	

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <get-authenticitycheckrule output="/tmp/MyApp-authenticitycheckrule.txt"/>
</app-version>
```

This command is based on the "Application (GET)" on page 1046 REST service.

The <set-authenticitycheckrule> command

The <set-authenticitycheckrule> command changes the authenticity check rule for an app version. It has the following attributes:

Table 261. <set-authenticitycheckrule> command attributes

Attribute	Description	Required	Default
action	Action to perform for authenticity checking	Yes	

The possible actions are:
- DISABLED: Authenticity is not checked.
- IGNORED: Authenticity is checked, but not enforced. If it fails, only a warning is given and the session is authorized.
- ENABLED: Authenticity is checked and enforced.

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <set-authenticitycheckrule action="enabled"/>
</app-version>
```

This command is based on the "App Version Authenticity Check (PUT)" on page 1033 REST service.

The <get-lock> command

The <get-lock> command returns information about whether a app version is locked or unlocked. It has the following attributes:

Table 262. <get-lock> command attributes

Attribute	Description	Required	Default
output	Name of a file in which to store the output.	No	
outputproperty	Name of an Ant property in which to store the output.	No	

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <get-lock output="/tmp/MyApp-lock.txt"/>
</app-version>
```

This command is based on the "Application (GET)" on page 1046 REST service.

The <set-lock> command

The <set-lock> command sets an app version to locked or unlocked state. It has the following attributes:

Table 263. <get-lock> command attributes

Attribute	Description	Required	Default
lock	New lock state.	Yes	

The possible lock values are true and false.

Example:

```
<app-version runtime="worklight" name="MyApp" environment="android" version="1.1">
  <set-lock lock="true"/>
</app-version>
```

This command is based on the "App Version Lock (PUT)" on page 1040 REST service.

Commands for devices

When you invoke the <wladm> Ant task, you can include various commands for devices.

The <list-devices> command

The <list-devices> command returns the list of devices that have contacted the apps of a runtime. It has the following attributes:

Table 264. <list-devices> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
query	A friendly name or user identifier, to search for	No	
output	Name of output file	No	

Table 264. <list-devices> command attributes (continued)

Attribute	Description	Required	Default
outputproperty	Name of Ant property for the output	No	

The `query` parameter specifies a string to search for. All devices that have a friendly name or user identifier that contains this string (with case-insensitive matching) are returned.

Examples:

```
<list-devices runtime="worklight"/>
```

```
<list-devices runtime="worklight" query="john"/>
```

This command is based on the "Devices (GET)" on page 1068 REST service.

The <remove-device> command

The `<remove-device>` command clears the record about a device that has contacted the apps of a runtime. It has the following attributes:

Table 265. <remove-device> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
id	Unique device identifier	Yes	

Example:

```
<remove-device runtime="worklight" id="496E974CCEDE86791CF9A8EF2E5145B6"/>
```

This command is based on the "Device (DELETE)" on page 1061 REST service.

The <device> command group

The `<device>` command group has the following attributes:

Table 266. <device> command group attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
id	Unique device identifier	Yes	

It supports the following elements:

Table 267. <device> command group elements

Element	Description	Count
set-status	Changes the status	0..∞

Table 267. <device> command group elements (continued)

Element	Description	Count
set-appstatus	Changes the status for an app	0..∞

The <set-status> command

The <set-status> command changes the status of a device, in the scope of a runtime. It has the following attributes:

Table 268. <set-status> command attributes

Attribute	Description	Required	Default
status	New status.	Yes	

The status can be one of::
- ACTIVE
- LOST
- STOLEN
- EXPIRED
- DISABLED

Example:
```
<device runtime="worklight" id="496E974CCEDE86791CF9A8EF2E5145B6">
  <set-status status="EXPIRED"/>
</device>
```

This command is based on the "Device Status (PUT)" on page 1064 REST service.

The <set-appstatus> command

The <set-appstatus> command changes the status of a device, regarding an app in a runtime. It has the following attributes:

Table 269. <set-appstatus> command attributes

Attribute	Description	Required	Default
app	Name of an app.	Yes	
status	New status.	Yes	

The status can be one of::
- ENABLED
- DISABLED

Example:
```
<device runtime="worklight" id="496E974CCEDE86791CF9A8EF2E5145B6">
  <set-appstatus app="MyApp" status="DISABLED"/>
</device>
```

This command is based on the "Device Application Status (PUT)" on page 1058 REST service.

Commands for troubleshooting

The following commands can help investigate problems with the Worklight Server web applications.

The <show-info> command

The `<show-info>` command shows basic information about the Worklight administration services, that can be returned without accessing any runtime nor database. This command can be used to test whether the Worklight administration services are running at all. It has the following attributes:

Table 270. <show-info> command attributes

Attribute	Description	Required	Default
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<show-info/>
```

The <show-versions> command

The `<show-versions>` command displays the Worklight versions of various components:

- `wladmVersion`: the exact Worklight Server version number from which `worklight-ant-deployer.jar` is taken.
- `productVersion`: the exact Worklight Server version number from which `worklightadmin.war` is taken.

And for every project WAR file:

- `serverVersion`: the exact Worklight Server version number from which `worklight-jee-library.jar` is taken.
- `platformVersion`: the exact version number of the Worklight development tools (Studio) that built the project WAR file.

It has the following attributes:

Table 271. <show-versions> command attributes

Attribute	Description	Required	Default
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Example:

```
<show-versions/>
```

The <list-runtimes> command

The <list-runtimes> command returns a list of the deployed runtimes (Worklight projects). It has the following attributes:

Table 272. <list-runtimes> command attributes

Attribute	Description	Required	Default
inDatabase	Whether to look in the database instead of via MBeans	No	false
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Examples:

```
<list-runtimes/>
```

```
<list-runtimes inDatabase="true"/>
```

This command is based on the "Runtimes (GET)" on page 1089 REST service.

The <show-runtime> command

The <show-runtime> command shows information about a given deployed runtime (Worklight project). It has the following attributes:

Table 273. <show-runtime> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	
output	Name of output file	No	
outputproperty	Name of Ant property for the output	No	

Examples:

```
<show-runtime runtime="worklight"/>
```

This command is based on the "Runtime (GET)" on page 1080 REST service.

The <delete-runtime> command

The <delete-runtime> command deletes the runtime, including its apps and adapters, from the database. It is only possible to delete a runtime when its web application is stopped. It has the following attributes:

Table 274. <delete-runtime> command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / Worklight project.	Yes	

Table 274. <delete-runtime> command attributes (continued)

Attribute	Description	Required	Default
condition	Condition when to delete it: empty or always (dangerous!)	No	

Example:

```
<delete-runtime runtime="worklight" condition="empty"/>
```

This command is based on the "Runtime (DELETE)" on page 1079 REST service.

A complex example of a wladm Ant task

Here is an example of how to use several <wladm> invocations and XML processing to solve more complex tasks.

This example lists all access rules of all apps in all runtimes. The third-party XMLTask Ant task, from oopsconsultancy is used, which provides, in particular:

- Iteration over a list of XML elements that are specified through an XPath expression.
- Access to several attributes of an XML element, in each iteration.

Here is an example of the code:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project default="main">
  <!-- Prerequisite for using the <wladm> Ant task. -->
  <taskdef resource="com/worklight/ant/deployers/antlib.xml">
    <classpath>
      <pathelement location="/opt/IBM/Worklight/WorklightServer/worklight-ant-deployer.jar"/>
    </classpath>
  </taskdef>
  <!-- Prerequisite for using the <xmltask> Ant task. -->
  <taskdef name="xmltask" classname="com.oopsconsultancy.xmltask.ant.XmlTask">
    <classpath>
      <pathelement location="/opt/xmltask/xmltask.jar"/>
    </classpath>
  </taskdef>

  <!-- Parameters for every <wladm> invocation. -->
  <property name="url" value="https://localhost:8443/worklightadmin"/>
  <property name="user" value="demo"/>
  <property name="password" value="demo"/>
  <property name="secure" value="false"/>

  <target name="main">
    <wladm url="${url}" user="${user}" password="${password}" secure="${secure}">
      <list-runtimes output="/tmp/ListRuntimes.xml" inDatabase="true"/>
    </wladm>
    <xmltask source="/tmp/ListRuntimes.xml">
      <call path="/projectconfiguration/projects/project">
        <param name="runtime" path="@name"/>
        <actions>
          <echo message="runtime=@{runtime}"/>
          <sequential>
            <wladm url="${url}" user="${user}" password="${password}" secure="${secure}">
              <list-apps runtime="@{runtime}" output="/tmp/ListApps.xml"/>
            </wladm>
            <xmltask source="/tmp/ListApps.xml">
              <call path="/applications/items/item/appVersions/appVersion">
                <param name="name" path="@application"/>
                <param name="environment" path="@environment"/>
```

```
                          <param name="version" path="@version"/>
                          <actions>
                            <sequential>
                              <echo message="Access rules for app name=@{name}, environment=@{environment},
                              <wladm url="${url}" user="${user}" password="${password}" secure="${secure}">
                                <app-version runtime="@{runtime}" name="@{name}" environment="@{environment}
                                  <get-accessrule/>
                                </app-version>
                              </wladm>
                            </sequential>
                          </actions>
                        </call>
                      </xmltask>
                    </sequential>
                  </actions>
                </call>
              </xmltask>
            </target>
        </project>
```

Administering Worklight applications through the command line

You can administer Worklight applications through the `wladm` program.

Comparison with other facilities

You can execute administration operations with IBM Worklight Foundation
through:
- The Worklight Console, which is interactive.
- The <wladm> Ant task.
- The `wladm` program.
- The Worklight administration REST services.

The last three are useful for automated or unattended execution of operations, for
example:
- to eliminate operator errors for repetitive operations
- to execute the operations outside the operator's normal working hours

The `wladm` program and the <wladm> Ant task are simpler to use and have better
error reporting than the REST services. The advantage of the `wladm` program over
the <wladm> is that it is easier to integrate when an integration with operating
system commands is already available. Additionally, it is more suitable to
interactive use.

Prerequisites

The `wladm` command is provided in the *WL_INSTALL_DIR*/shortcuts/ directory as a
set of scripts:
- `wladm` for UNIX / Linux
- `wladm.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.

In order to use the `wladm` program, you need to either put the
WL_INSTALL_DIR/shortcuts/ directory into your PATH environment variable, or

reference its absolute file name in each invocation.

Invoking the wladm program

You can use the `wladm` program to administer Worklight applications.

Invocation

The `wladm` program is invoked as follows:

```
wladm --url= --user= ... [--passwordfile=...] [--secure=false] some command
```

The `wladm` program has the following options:

Table 275. wladm program options

Option	Type	Description	Required	Default
--url	URL	Base URL of the Worklight admin services web application	Yes	
--secure	boolean	Whether to avoid operations with security risks	No	true
--user	name	User name for accessing the Worklight admin services	Yes	
--passwordfile	file	File containing the password for the user	No	
--verbose		Detailed output	No	

The URL is the base URL of the Worklight admin services web application, that preferably uses the HTTPS protocol. For example, if you use default ports and context roots, this URL is:

- `https://server:9443/worklightadmin` for a WebSphere server.
- `https://server:8443/worklightadmin` for a Tomcat server.

The `--secure` option is set to `true` by default. If you use `--secure=false`:

- The user and password might be transmitted in an insecure way (possibly even through unencrypted HTTP).
- The server's SSL certificates are accepted even if self-signed or if they were created for a different hostname from the server's hostname.

Specify the password in a separate file, indicated through the `--passwordfile` option. In interactive mode (see "Interactive mode" on page 1278), you can alternatively specify the password interactively. The password is sensitive information and therefore needs to be protected. You must prevent other users on the same computer from learning these passwords. To secure the password, before you enter the password into a file, you must remove the read permissions of the file for users other than yourself. For example, you can use one of the following commands:

- On UNIX: chmod 600 *adminpassword.txt*

- On Windows: cacls *adminpassword.txt* /P Administrators:F
  %USERDOMAIN%\%USERNAME%:F

For this reason, it is also not recommended to pass the password to a process through a command-line argument. On many operating systems, other users can inspect the command-line arguments of your processes.

The wladm invocation contains a command. The following commands are supported:

Table 276. wladm invocation supported commands

Command	Description
show info	Shows user and configuration information
show versions	Shows version information
list runtimes [--in-database]	Lists the runtimes
show runtime [*runtime-name*]	Shows information about a runtime
delete runtime [*runtime-name*] *condition*	Deletes a runtime
list adapters [*runtime-name*]	Lists the adapters
deploy adapter [*runtime-name*] *file*	Deploys an adapter
show adapter [*runtime-name*] *adapter-name*	Shows information about an adapter
delete adapter [*runtime-name*] *adapter-name*	Deletes an adapter
adapter [*runtime-name*] *adapter-name* get binary [> *tofile*]	Get the binary data of an adapter
list apps [*runtime-name*]	Lists the apps
deploy app [*runtime-name*] *file*	Deploys an app
show app [*runtime-name*] *app-name*	Shows information about an app
delete app [*runtime-name*] *app-name*	Deletes an app
delete app version [*runtime-name*] *app-name environment version*	Deletes a version of an app
app version [*runtime-name*] *app-name environment version* get binary [> *tofile*]	Gets the binary data of an app version
app version [*runtime-name*] *app-name environment version* get accessrule	Gets the access rule of an app version
app version [*runtime-name*] *app-name environment version* set accessrule *file*	Changes the access rule of an app version
app version [*runtime-name*] *app-name environment version* get authenticitycheckrule	Gets the authenticity check rule of an app version
app version [*runtime-name*] *app-name environment version* set authenticitycheckrule *action*	Changes the authenticity check rule of an app version
app version [*runtime-name*] *app-name environment version* get lock	Gets the lock state of an app version
app version [*runtime-name*] *app-name environment version* set lock *lock*	Changes the lock state of an app version
list devices [*runtime-name*] [--query *query]*	Lists the devices

Table 276. wladm invocation supported commands (continued)

Command	Description
remove device [*runtime-name*] *id*	Removes a device
device [*runtime-name*] *id* set status *new-status*	Changes the status of a device
device [*runtime-name*] *id* set appstatus *app-name new-status*	Changes the status of a device for an app

Interactive mode

Alternatively, you can also call wladm without any command that is specified in the command line. You can then enter commands interactively, one per line.

The command **exit**, or end-of-file on standard input (**Ctrl-D** on UNIX terminals) terminates wladm.

Help commands are available in this mode as well. For example:
- **help**
- **help show versions**
- **help device**
- **help device set status**

Command history in interactive mode

On some operating systems, the interactive wladm command remembers the command history. With the command history, you can select a previous command, using the arrow-up and arrow-down keys, edit it, and execute it.
- On Linux, the command history is enabled in terminal emulator windows if the rlwrap package is installed and found in PATH. To install the rlwrap package:
  - On Red Hat Linux: **sudo yum install rlwrap**
  - On SUSE Linux: **sudo zypper install rlwrap**
  - On Ubuntu: **sudo apt-get install rlwrap**
- On OS X, the command history is enabled in the Terminal program if the rlwrap package is installed and found in PATH. To install the rlwrap package:
  1. Install MacPorts by using the installer from www.macports.org.
  2. Run the command:

     sudo /opt/local/bin/port install rlwrap

     Then, to make the rlwrap program available in PATH, use this command in a Bourne-compatible shell:

     PATH=/opt/local/bin:$PATH
- On Windows, the command history is enabled in cmd.exe console windows.

In environments where rlwrap does not work or is not desired, you can disable its use through the option **--no-readline**.

The configuration file

As a convenience, it is also possible to store the options in a configuration file, instead of passing them on the command line at every invocation. The options that can be omitted when a configuration file is present and the option –configfile=file is specified are:

- `--url=`*URL*
- `--secure=`*boolean*
- `--user=`*name*
- `--passwordfile=`*file*
- *runtime-name*

The commands to store these values in the configuration file are:

Table 277. Commands to store values in the configuration file

Command	Comment
`wladm [--configfile=`*file*`] config url` *URL*	
`wladm [--configfile=`*file*`] config secure` *boolean*	
`wladm [--configfile=`*file*`] config user` *name*	
`wladm [--configfile=`*file*`] config password`	Prompts for the password.
`wladm [--configfile=`*file*`] config runtime` *runtime-name*	

The command to list the values that are stored in the configuration file is: `wladm [--configfile=`*file*`] config` .

The configuration file is a text file, in the encoding of the current locale, in Java `.properties` syntax. The default configuration file is on

- Unix: `$HOME/.wladm.config`
- Windows: `My Documents\IBM Worklight Server Data\wladm.config`

Note: When you do not specify a `--configfile` option, the default configuration file is only used in interactive mode and in `config` commands. For non-interactive use of the other commands, you must explicitly designate the configuration file if you want to use one.

Note: The password is stored in an obfuscated format that hides the password from an occasional glimpse. However, this obfuscation provides no security.

Generic options

There are also the usual generic options:

Table 278. Generic options

Option	Description
`--help`	Shows some usage help
`--version`	Shows the version

XML format

The commands that receive an XML response from the server verify that this response obeys the specific schema. You can disable this check by specifying an option `--xmlvalidation=none`.

Output character set

Normal output that is produced by the `wladm` program is encoded in the encoding of the current locale. On Windows, this encoding is the so-called "ANSI code page". The effects are as follows:

- Characters outside of this character set are converted to question marks when they are output.
- When a piece of XML is being output, it is still labeled as `<?xml version="1.0" encoding="UTF-8"?>` although in reality the encoding of the output is the encoding of the current locale.
- When the output goes to a Windows command prompt (`cmd.exe`) window, non-ASCII characters are incorrectly displayed because such windows assume characters to be encoded in the so-called "OEM code page".

To work around this limitation:

- On operating systems other than Windows, use a locale whose encoding is UTF-8. This is the default locale on Red Hat Linux and OS X; many other operating systems have a locale that is named `en_US.UTF-8`.
- Or use the `<wladm>` Ant task, with an attribute `output="`*some file name*`"` to redirect the output of a command to a file.

Commands for adapters

When you invoke the `wldam` program, you can include various commands for adapters.

The list adapters command

The `list adapters` command returns a list of the adapters that are deployed for a runtime.

Syntax:

```
list adapters [runtime-name]
```

It takes the following arguments:

Table 279. list adapters command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.

It takes the following options after the object:

Table 280. list adapters options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:
```
list adapters worklight
```

This command is based on the "Adapters (GET)" on page 1026 REST service.

The deploy adapter command

The `deploy adapter` command deploys an adapter in a runtime.

Syntax:
```
deploy adapter [runtime-name] file
```

It takes the following arguments:

Table 281. deploy adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
file	Binary adapter file (.adapter)

Example:
```
deploy adapter worklight MyAdapter.adapter
```

This command is based on the "Adapter (POST)" on page 1022 REST service.

The show adapter command

The `show adapter` command shows details about an adapter.

Syntax:
```
show adapter [runtime-name] adapter-name
```

It takes the following arguments:

Table 282. show adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
adapter-name	Name of an adapter

It takes the following options after the object:

Table 283. show adapter options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:
```
show adapter worklight MyAdapter
```

This command is based on the "Adapter (GET)" on page 1020 REST service.

The delete adapter command

The `delete adapter` command removes (undeploys) an adapter from a runtime.

Syntax:

```
delete adapter [runtime-name] adapter-name
```

It takes the following arguments:

Table 284. delete adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
adapter-name	Name of an adapter

Example:

```
delete adapter worklight MyAdapter
```

This command is based on the "Adapter (DELETE)" on page 1017 REST service.

The adapter command prefix

The `adapter` command prefix takes the following arguments before the verb:

Table 285. adapter command prefix arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
adapter-name	Name of an adapter

The adapter get binary command

The `adapter get binary` command returns the binary adapter file.

Syntax:

```
adapter [runtime-name] adapter-name get binary [> tofile]
```

It takes the following options after the verb:

Table 286. adapter get binary options

Option	Description	Required	Default
> tofile	Name of the output file.	No	Standard output

Example:

```
adapter worklight MyAdapter get binary > /tmp/MyAdapter.adapter
```

This command is based on the "Adapter Binary (GET, HEAD)" on page 1016 REST service.

Commands for apps

When you invoke the `wladm` program, you can include various commands for apps.

The list apps command

The `list apps` command returns a list of the apps that are deployed in a runtime.

Syntax:

```
list apps [runtime-name]
```

It takes the following arguments:

Table 287. list apps command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.

It takes the following options after the object:

Table 288. list apps options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
list apps worklight
```

This command is based on the "Applications (GET)" on page 1054 REST service.

The deploy app command

The `deploy app` command deploys an app (possibly with multiple environments) in a runtime.

Syntax:

```
deploy app [runtime-name] file
```

It takes the following arguments:

Table 289. deploy app command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
file	Binary app file (.wlapp, not .apk or .ipa)

Example:

```
deploy app worklight MyApp-all.wlapp
```

This command is based on the "Application (POST)" on page 1050 REST service.

The show app command

The `show app` command shows details about an app in a runtime, in particular its environments and versions.

Syntax:

```
show app [runtime-name] app-name
```

It takes the following arguments:

Table 290. show app command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
app-name	Name of an app

It takes the following options after the object:

Table 291. show app options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:
```
show app worklight MyApp
```

This command is based on the "Application (GET)" on page 1046 REST service.

The delete app command

The delete app command removes (undeploys) an app (from all environments, and all versions) from a runtime.

Syntax:
```
delete app [runtime-name] app-name
```

It takes the following arguments:

Table 292. delete app command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
app-name	Name of an app

Example:
```
delete app worklight MyApp
```

This command is based on the "Application (DELETE)" on page 1043 REST service.

The delete app version command

The delete app version command removes (undeploys) an app version from a runtime.

Syntax:
```
delete app version [runtime-name] app-name environment version
```

It takes the following arguments:

Table 293. delete app version command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
app-name	Name of an app
environment	Mobile platform
version	Version of the app

Example:

```
delete app version worklight MyApp android 1.1
```

This command is based on the "App Version (DELETE)" on page 1037 REST service.

The app version command prefix

The `app version` command prefix takes the following arguments before the verb:

Table 294. app version command prefix arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
app-name	Name of an app
environment	Mobile platform
version	Version of the app

The app version get binary command

The `app version get binary` command returns the binary `wlapp` file for a version of an app.

Syntax:

```
app version [runtime-name] app-name environment version get binary [> tofile]
```

It takes the following arguments after the verb:

Table 295. delete app version command arguments

Argument	Description	Required	Default
> tofile	Name of the output file.	No	Standard output

Example:

```
app version worklight MyApp android 1.1 get binary > /tmp/MyApp.wlapp
```

This command is based on the "Application Binary (GET, HEAD)" on page 1042 REST service.

The app version get accessrule command

The `app version get accessrule` command returns the access rule for an app version.

Syntax:

`app version [`*`runtime-name`*`] ` *`app-name environment version`* ` get accessrule`

Example:

`app version worklight MyApp android 1.1 get accessrule`

This command is based on the "Application (GET)" on page 1046 REST service.

The app version set accessrule command

The `app version set accessrule` command changes the access rule for an app version.

Syntax:

`app version [`*`runtime-name`*`] ` *`app-name environment version`* ` set accessrule ` *`file`*

It takes the following arguments after the verb:

Table 296. app version set accessrule command arguments

Argument	Description
`file`	Name of the input file.

Example:

`app version worklight MyApp android 1.1 set accessrule /tmp/new-accessrule.xml`

This command is based on the "App Version Access Rule (PUT)" on page 1028 REST service.

The app version get authenticitycheckrule command

The `app version get authenticitycheckrule` command returns the authenticity check rule for an app version.

Syntax:

`app version [`*`runtime-name`*`] ` *`app-name environment version`* ` get authenticitycheckrule`

Example:

`app version worklight MyApp android 1.1 get authenticitycheckrule`

This command is based on the "Application (GET)" on page 1046 REST service.

The app version set authenticitycheckrule command

The `app version set authenticitycheckrule` command changes the authenticity check rule for an app version.

Syntax:

`app version [`*`runtime-name`*`] ` *`app-name environment version`* ` set authenticitycheckrule ` *`action`*

It takes the following arguments after the verb:

Table 297. app version set authenticitycheckrule command arguments

Argument	Description
action	Action to perform for authenticity checking

The possible actions are:

- DISABLED: Authenticity is not checked
- IGNORED: Authenticity is checked, but not enforced. If it fails, only a warning is given and the session is authorized
- ENABLED: Authenticity is checked and enforced

Example:

```
app version worklight MyApp android 1.1 set authenticitycheckrule enabled
```

This command is based on the "App Version Authenticity Check (PUT)" on page 1033 REST service.

The app version get lock command

The app version get lock command returns information about whether an app version is locked or unlocked.

Syntax:

```
app version [runtime-name] app-name environment version get lock
```

Example:

```
app version worklight MyApp android 1.1 get lock
```

This command is based on the "Application (GET)" on page 1046 REST service.

The app version set lock command

The app version set lock command sets an app version to locked or unlocked state.

Syntax:

```
app version [runtime-name] app-name environment version set lock lock
```

It takes the following arguments after the verb:

Table 298. app version set lock command arguments

Argument	Description
lock	New lock state.

The possible lock values are true and false.

Example:

```
app version worklight MyApp android 1.1 set lock true
```

This command is based on the "App Version Lock (PUT)" on page 1040 REST service.

Commands for devices

When you invoke the `wladm` program, you can include various commands for devices.

The list devices command

The `list devices` command returns the list of devices that have contacted the apps of a runtime.

Syntax:

```
list devices [runtime-name] [--query query]
```

It takes the following arguments:

Table 299. list devices command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
query	A friendly name or user identifier, to search for

The `query` parameter specifies a string to search for. All devices that have a friendly name or user identifier that contains this string (with case-insensitive matching) are returned.

It takes the following options after the object:

Table 300. list devices options

Option	Description
--xml	Produce XML output instead of tabular output.

Examples:

```
list-devices worklight
list-devices worklight --query=john
```

This command is based on the "Devices (GET)" on page 1068 REST service.

The remove device command

The `remove device` command clears the record about a device that has contacted the apps of a runtime.

Syntax:

```
remove device [runtime-name] id
```

It takes the following arguments:

Table 301. remove device command arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
id	Unique device identifier

Example:
```
remove device worklight 496E974CCEDE86791CF9A8EF2E5145B6
```

This command is based on the "Device (DELETE)" on page 1061 REST service.

The device command prefix

The `device` command prefix takes the following arguments before the verb:

Table 302. device command prefix arguments

Argument	Description
`runtime-name`	Name of the runtime web application / Worklight project.
`id`	Unique device identifier

The device set status command

The `device set status` command changes the status of a device, in the scope of a runtime.

Syntax:
```
device [runtime-name] id set status new-status
```

It takes the following arguments:

Table 303. device set status command arguments

Argument	Description
`new-status`	New status.

The status can be:
- `ACTIVE`
- `LOST`
- `STOLEN`
- `EXPIRED`
- `DISABLED`

Example:
```
device worklight 496E974CCEDE86791CF9A8EF2E5145B6 set status EXPIRED
```

This command is based on the "Device Status (PUT)" on page 1064 REST service.

The device set appstatus command

The `device set appstatus` command changes the status of a device, regarding an app in a runtime.

Syntax:
```
device [runtime-name] id set appstatus app-name new-status
```

It takes the following arguments:

Table 304. device set appstatus command arguments

Argument	Description
app-name	Name of an app.
new-status	New status.

The status can be:

- ENABLED
- DISABLED

Example:

```
device worklight 496E974CCEDE86791CF9A8EF2E5145B6 set appstatus MyApp DISABLED
```

This command is based on the "Device Application Status (PUT)" on page 1058 REST service.

Commands for troubleshooting

When you invoke the wladm program, you can include various commands for troubleshooting.

The show info command

The show info command shows basic information about the Worklight administration services that can be returned without accessing any runtime nor database. This command can be used to test whether the Worklight administration services are running at all.

Syntax:

```
show info
```

It takes the following options after the object:

Table 305. show info options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
show info
```

The show versions command

The show versions command displays the Worklight versions of various components:

- wladmVersion: the exact Worklight Server version number from which worklight-ant-deployer.jar is taken.
- productVersion: the exact Worklight Server version number from which worklightadmin.war is taken

and for every project WAR file:

- serverVersion: the exact Worklight Server version number from which worklight-jee-library.jar is taken
- platformVersion: the exact version number of the Worklight development tools (Studio) that built the project WAR file

Syntax:
```
show versions
```

It takes the following options after the object:

Table 306. show versions options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:
```
show versions
```

The list runtimes command

The `list runtimes` command returns a list of the deployed runtimes (Worklight projects).

Syntax:
```
list runtimes [--in-database]
```

It takes the following options:

Table 307. list runtimes options

Option	Description
--in-database	Whether to look in the database instead of via MBeans
--xml	Produce XML output instead of tabular output.

Examples:
```
list runtimes
list runtimes --in-database
```

This command is based on the "Runtimes (GET)" on page 1089 REST service.

The show runtime command

The `show runtime` command shows information about a given deployed runtime (Worklight project).

Syntax:
```
show runtime [runtime-name]
```

It takes the following arguments:

Table 308. show runtime arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.

It takes the following options after the object:

Table 309. show runtime options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
show runtime worklight
```

This command is based on the "Runtime (GET)" on page 1080 REST service.

The delete runtime command

The `delete runtime` command deletes a runtime, including its apps and adapters, from the database. It is only possible to delete a runtime when its web application is stopped.

Syntax:

```
delete runtime [runtime-name] condition
```

It takes the following arguments:

Table 310. delete runtime arguments

Argument	Description
runtime-name	Name of the runtime web application / Worklight project.
condition	Condition when to delete it: 'empty' or 'always' (dangerous!)

Example:

```
delete runtime worklight empty
```

This command is based on the "Runtime (DELETE)" on page 1079 REST service.

Administering push notifications with the Worklight Console

The Push Notifications page in the Worklight Console provides you with a quick view of the various entities in the push notification chain.
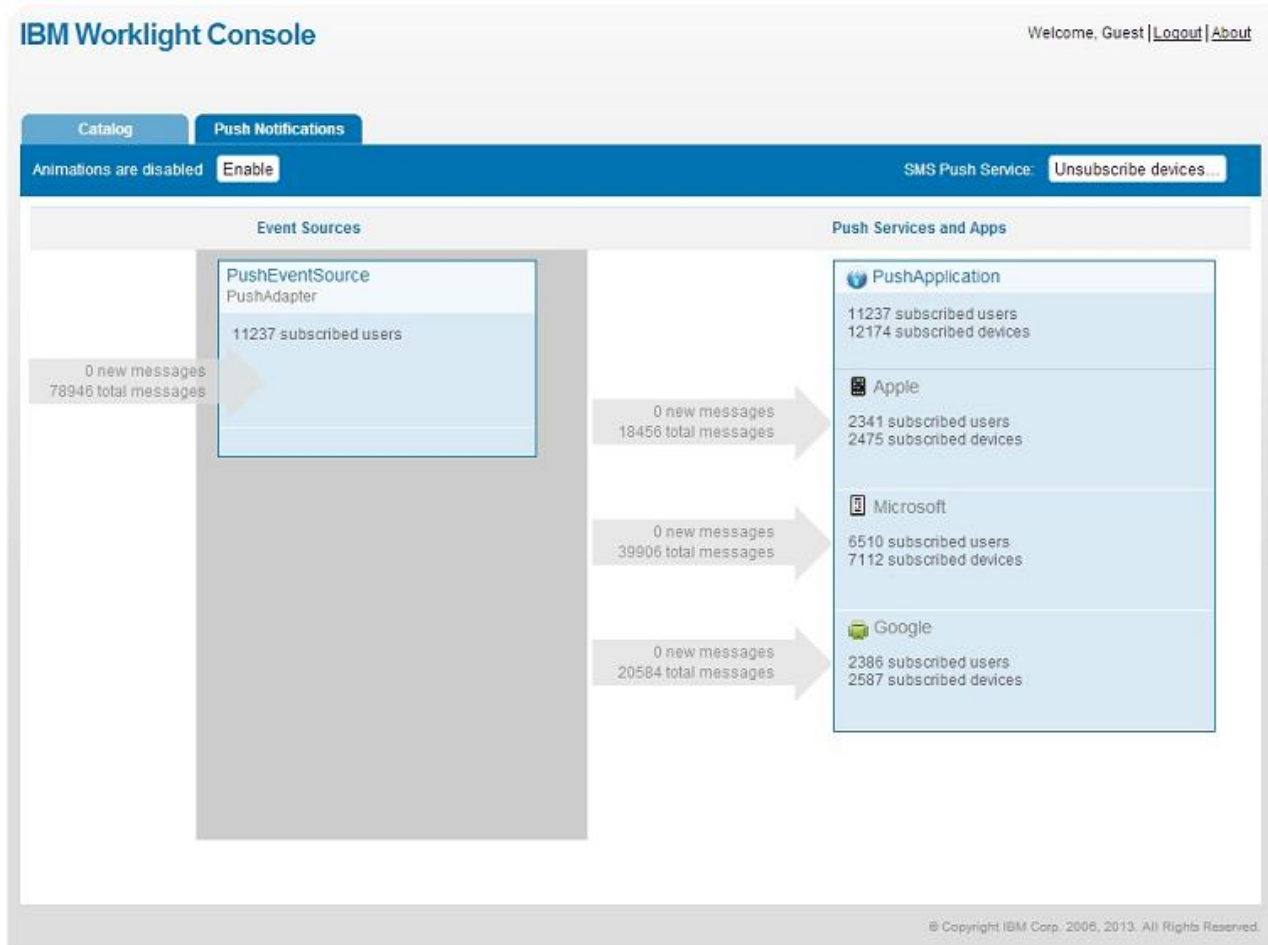
Figure 155. Push notifications in the Worklight Console

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.
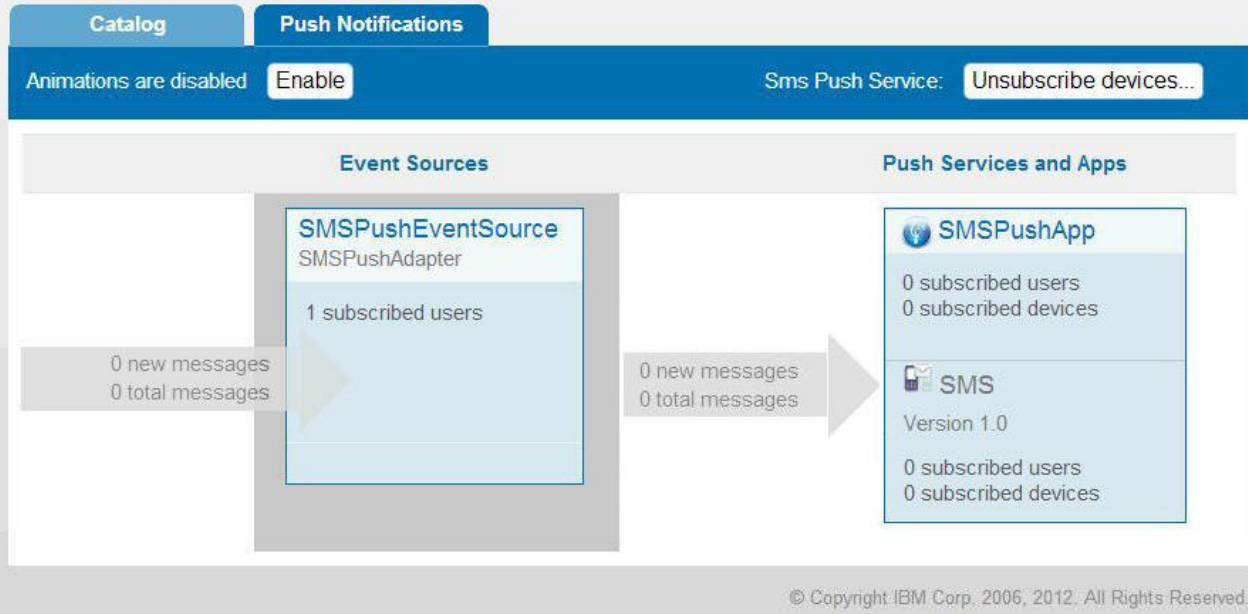
Figure 156. SMS push notifications in the Worklight Console

Administrators can forcibly unsubscribe existing SMS subscriptions by clicking **Unsubscribe devices**. The Unsubscribe SMS Devices window opens, and administrators can then enter the mobile phone numbers to be unsubscribed.

Note: It is possible to have two subscriptions for the same phone number and user name; one created by using the device and one created by using the subscribe SMS servlet. If there are two subscriptions for the same phone number and user name, unsubscription by using the Worklight Console unsubscribes both subscriptions.

Figure 157. 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 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 1340 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 1314.

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 1350.

The following figure shows an overview of the architecture.
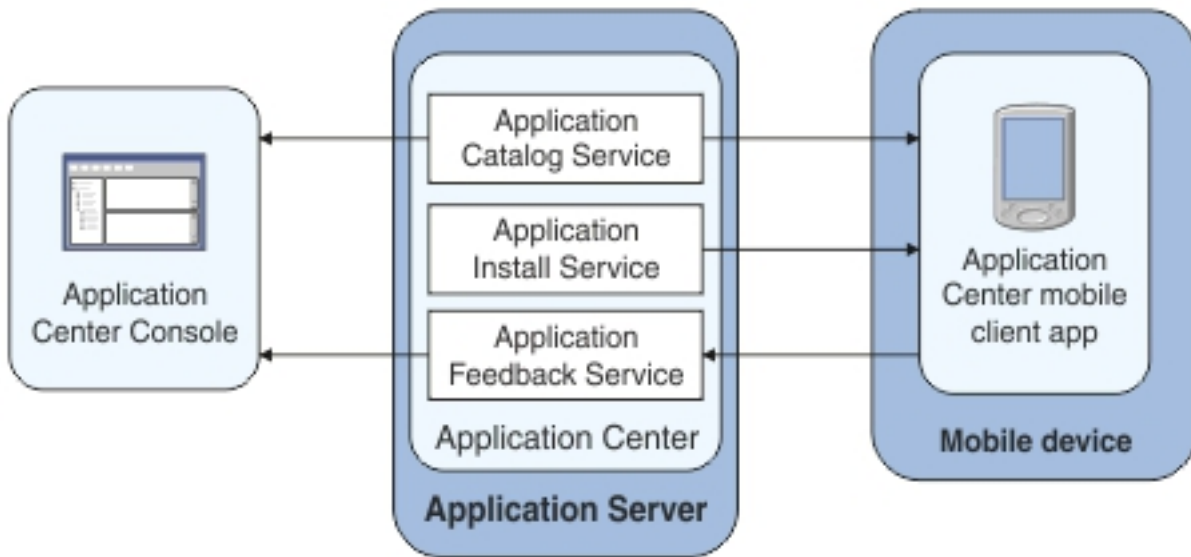
Figure 158. Architecture of the Application Center

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 Foundation 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 Foundation is installed, start the Application Center console, and log in.

When you install IBM Worklight Foundation, 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 264 or, if you are using LDAP authentication, "Managing users with LDAP" on page 270.

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 1314.

To install and run the mobile client on:
- Android operating system: see "Installing the client on an Android mobile device" on page 1350
- iOS operating system: see "Installing the client on an iOS mobile device" on page 1353
- BlackBerry OS 6 and OS 7: see "Installing the client on a BlackBerry mobile device" on page 1354.
- Windows Phone 8: see "Installing the client on Windows Phone 8" on page 1355

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 a 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 1309 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 1309 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 1309 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 Chapter 9, "Developing Worklight applications," on page 409 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 311. 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 311. 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.

Hybrid splash screen during auto login: /IBMAppCenter/apps/AppCenter/
common/js/idx/mobile/themes/common/idx/Launch.css

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.

Enabling high-resolution splash images for iPhone 6 and 6 Plus devices:

Use a **UILaunchImages** dictionary to enable high-resolution splash images.

During app startup on earlier versions of iPhone, iOS looked for splash images
among application resources. IBM Worklight looked for the same images and
displayed them during WebView initialization. For iPhone 6 and 6 Plus, however,
iOS is unable to automatically detect high-resolution images and IBM Worklight is
therefore not aware of them. To enable a high-resolution splash screen image on
iPhone 6 and 6 Plus, developers must create a dictionary that references the set of
images in the application resources. The dictionary comprises **UILaunchImages**
key-value pairs and must be located in the Info.plist file. When the reference to
the set appears in the Info.plist, both iOS and IBM Worklight are able to select
the splash image that is most suited to the device resolution.

The following code is a sample dictionary. To enable default Worklight splash
images, copy the code as-is into the Info.plist file to enable high-resolution
splash images for iPhone 6 or 6 Plus, along with low-resolution images for iPhone
3, 4, and 5.

Note: Images with lower resolutions must also appear in the dictionary.
The code uses the following image names:
- Default-667h.png: image for iPhone 6
- Default-736h.png: image for iPhone 6 Plus
- Default-736h-Landscape.png: image for iPhone 6 Plus in landscape orientation

Of course, you can customize the code to use your own images.

Note: The name of the splash image must be identical in both the dictionary and
the application resources.

```
<key>UILaunchImages</key>
<array>
    <dict>
        <key>UILaunchImageMinimumOSVersion</key>
```

```
            <string>8.0</string>
            <key>UILaunchImageName</key>
            <string>Default</string>
            <key>UILaunchImageOrientation</key>
            <string>Portrait</string>
            <key>UILaunchImageSize</key>
            <string>{320, 480}</string>
        </dict>
        <dict>
            <key>UILaunchImageMinimumOSVersion</key>
            <string>8.0</string>
            <key>UILaunchImageName</key>
            <string>Default-568h</string>
            <key>UILaunchImageOrientation</key>
            <string>Portrait</string>
            <key>UILaunchImageSize</key>
            <string>{320, 568}</string>
        </dict>
        <dict>
            <key>UILaunchImageMinimumOSVersion</key>
            <string>8.0</string>
            <key>UILaunchImageName</key>
            <string>Default-667h</string>
            <key>UILaunchImageOrientation</key>
            <string>Portrait</string>
            <key>UILaunchImageSize</key>
            <string>{375, 667}</string>
        </dict>
        <dict>
            <key>UILaunchImageMinimumOSVersion</key>
            <string>8.0</string>
            <key>UILaunchImageName</key>
            <string>Default-736h</string>
            <key>UILaunchImageOrientation</key>
            <string>Portrait</string>
            <key>UILaunchImageSize</key>
            <string>{414, 736}</string>
        </dict>
        <dict>
            <key>UILaunchImageMinimumOSVersion</key>
            <string>8.0</string>
            <key>UILaunchImageName</key>
            <string>Default-736h-Landscape</string>
            <key>UILaunchImageOrientation</key>
            <string>Landscape</string>
            <key>UILaunchImageSize</key>
            <string>{414, 736}</string>
        </dict>
    </array>
```

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 Application Center server.

Web This directory contains the packaged application for uploading over the air. This method is compatible with the 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 312. 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.
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 1317 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 1350
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 1353
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 1355
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.

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 sent to a device for applications that are marked as favorite. See "Marking or unmarking a favorite app" on page 1378. Push notifications are also sent to a device for applications that the administrator has marked as Recommended. See "Editing application properties" on page 1324.

Push notifications are currently not available for the Application Center client in BlackBerry or Windows Phone.

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.period.unit*.

The value of *ibm.appcenter.push.schedule.period.amount* is an integer. The value of *ibm.appcenter.push.schedule.period.unit* can be `seconds`, `minutes`, or `hours`. If the unit is not specified, the amount is an interval that is 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 297 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.String"
<Environment name="ibm.appcenter.push.schedule.period.amount" override="false" type="java.lang.String
```

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 1313 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
   // example : 123456789012
   ```

6. Register the Google API key as a JNDI property for the Application Center server. The key name is : **ibm.appcenter.gcm.signature.googleapikey**. For example, you can configure this key for an Apache Tomcat server as a JNDI property in the `server.xml` file:

   ```
   <Context docBase="AppCenterServices" path="/applicationcenter" reloadable="true" source="org.e
   <Environment name="ibm.appcenter.gcm.signature.googleapikey" override="false" type="java.lang.
   value="AIxaScCHg0VSGdgfOZKtzDJ44-oi0muUasMZvAs"/>
   </Context>
   ```

   The JNDI property must be defined in accordance with your application server requirements.

   See "List of JNDI properties for the Application Center" on page 297 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."
   - You must also ensure that your firewall accepts outgoing connections to `android.googleapis.com` on port 443 for push notifications to work.

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 313. JNDI properties

JNDI Property	Type and description	Example for Apache Tomcat server

`ibm.appcenter.apns.p12.certificate.location`	A String. location defines the full path to the `.p12` certificate.	`<Environment name="ibm.appcenter.apns.p12.certificate.loc`
		`override="false" type="java.lang.String" value=`
		`"/Users/someUser/someDirectory/apache-tomcat/conf/AppCente`

Table 313. JNDI properties (continued)

JNDI Property	Type and description	Example for Apache Tomcat server
`ibm.appcenter.apns.p12.certificate.password`	A string that defines the password needed to access the certificate.	`<Environment name="`*ibm.appcenter.apns.p12.certificate.p* `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.i` `override="false" type="java.lang.String"` `value="true"/>`

See "List of JNDI properties for the Application Center" on page 297 for a complete list of JNDI 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 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.

Chapter 13. Administering Worklight applications **1313**

7. Build the .apk file of the Application Center.

What to do next

The `gcm.jar` library is automatically added by the 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 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.

Figure 159. 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.

Figure 160. 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 1340 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**.

Figure 161. 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 1317.

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.
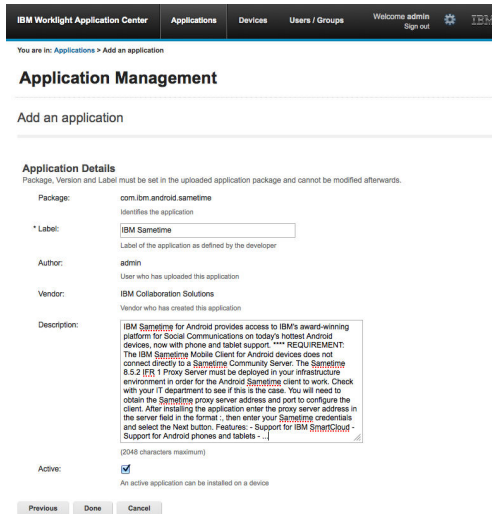
Figure 162. Modified application description in application properties

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.
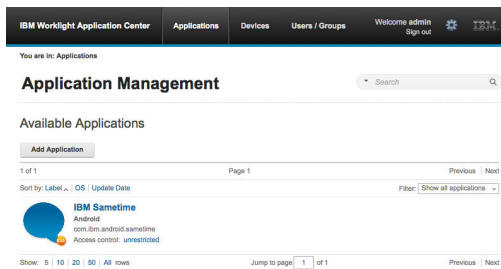
Figure 163. Link to an application stored in Google play

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 1370
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**.
- **External URL**; this property is supported for applications that run on Android, iOS, and Windows Phone 8.

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.
- **External URL** is a URL that enables you to have the mobile client of the Application Center launched automatically in the Details view of the latest version of the current application.

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.
- **External URL** is a URL that enables you to have the mobile client of the Application Center launched automatically in the Details view of the latest version of the current 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-l** 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.
- **External URL** is a URL that enables you to have the mobile client of the Application Center launched automatically in the Details view of the latest version of the current application.

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 1350.

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 1322 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.

Figure 164. Application properties for editing

Upgrading a mobile application in Worklight Server and the Application Center
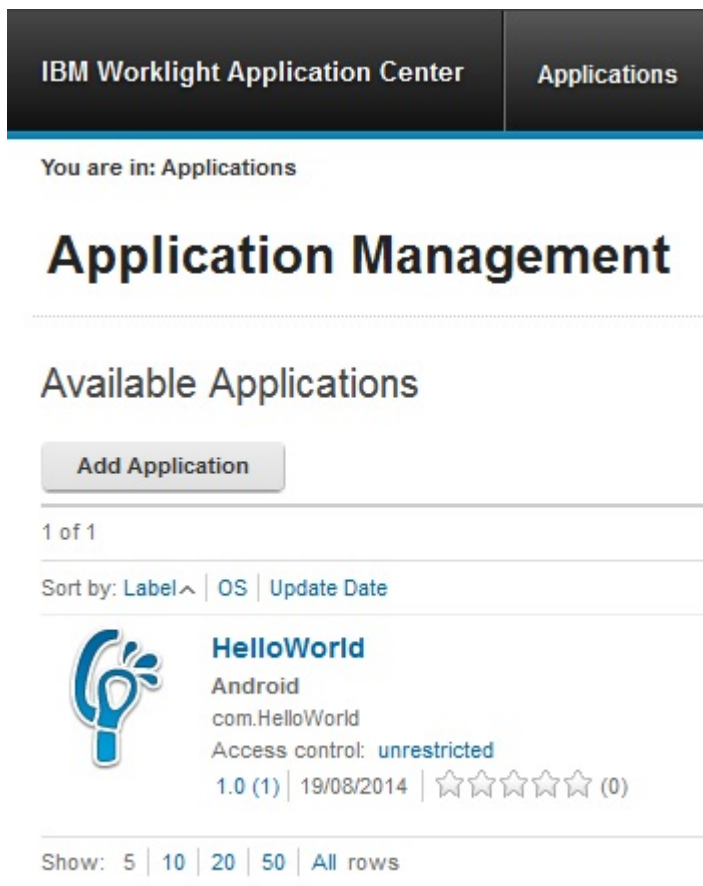
You can easily upgrade deployed mobile applications by using a combination of Worklight Console and the Application Center.

Before you begin

The mobile client of the Application Center must be installed on the mobile device. The HelloWorld application must be installed on the mobile device and must connect to Worklight Server when the application is running. See "Connecting to Worklight Server" on page 549.

About this task

You can use this procedure to update Android, iOS, and Windows Phone applications that have been deployed on Worklight Server and also in the Application Center. In this task, the application HelloWorld version 1.0 is already deployed on Worklight Server and in the Application Center.

Figure 165. Version 1.0 of the application is already deployed in Application Center
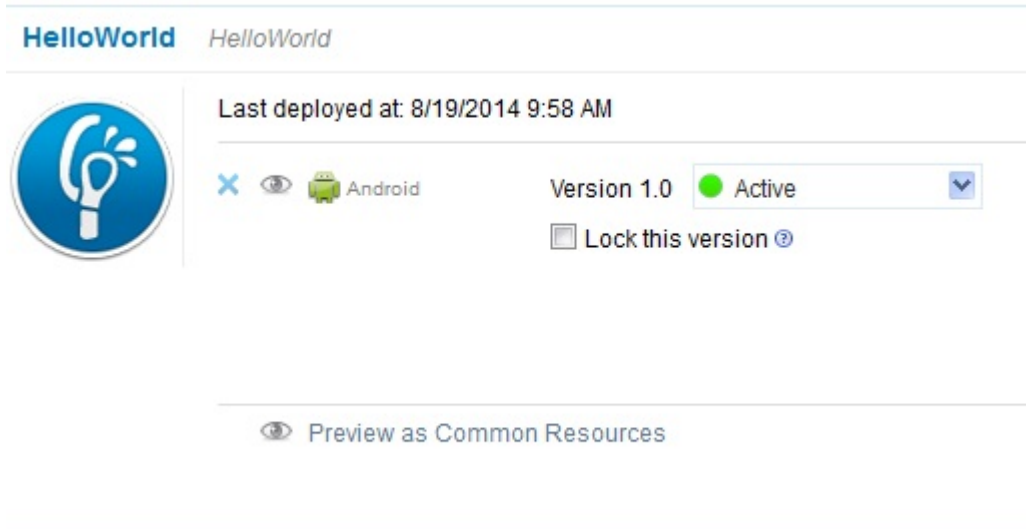
Figure 166. Version 1.0 of the application is already deployed on Worklight Server

Procedure

HelloWorld version 2.0 is released and you would like users of version 1.0 to upgrade to the later version. To deploy the new version of the application:

1. Deploy HelloWorld 2.0 in the Application Center. See "Adding a mobile application" on page 1317.
2. From the Application Details page, copy the setting of the external URL.

Figure 167. Copying the external URL from Application Details

3. When the external URL is copied to the clipboard, open the Worklight Console.
4. Change the access rule of HelloWorld version 1.0 to "Access Disabled".
5. Paste the external URL into the URL field.
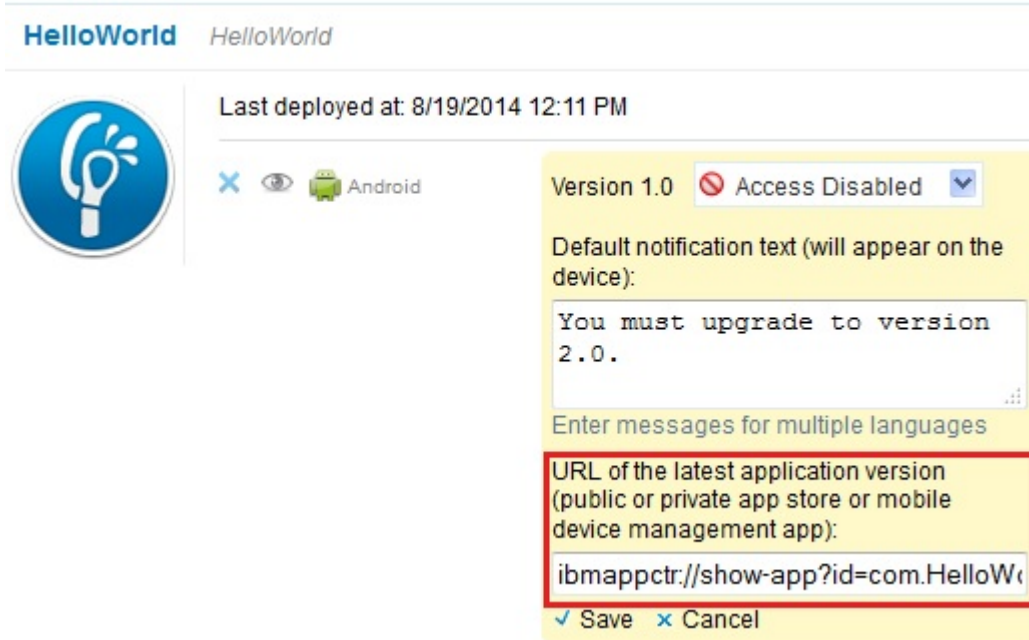
Figure 168. External URL pasted in Worklight Console with access disabled to the application version on Worklight Server

Running the client: When a mobile device connects to Worklight Server to try to run HelloWorld version 1.0, the device user is requested to upgrade the version of the application.
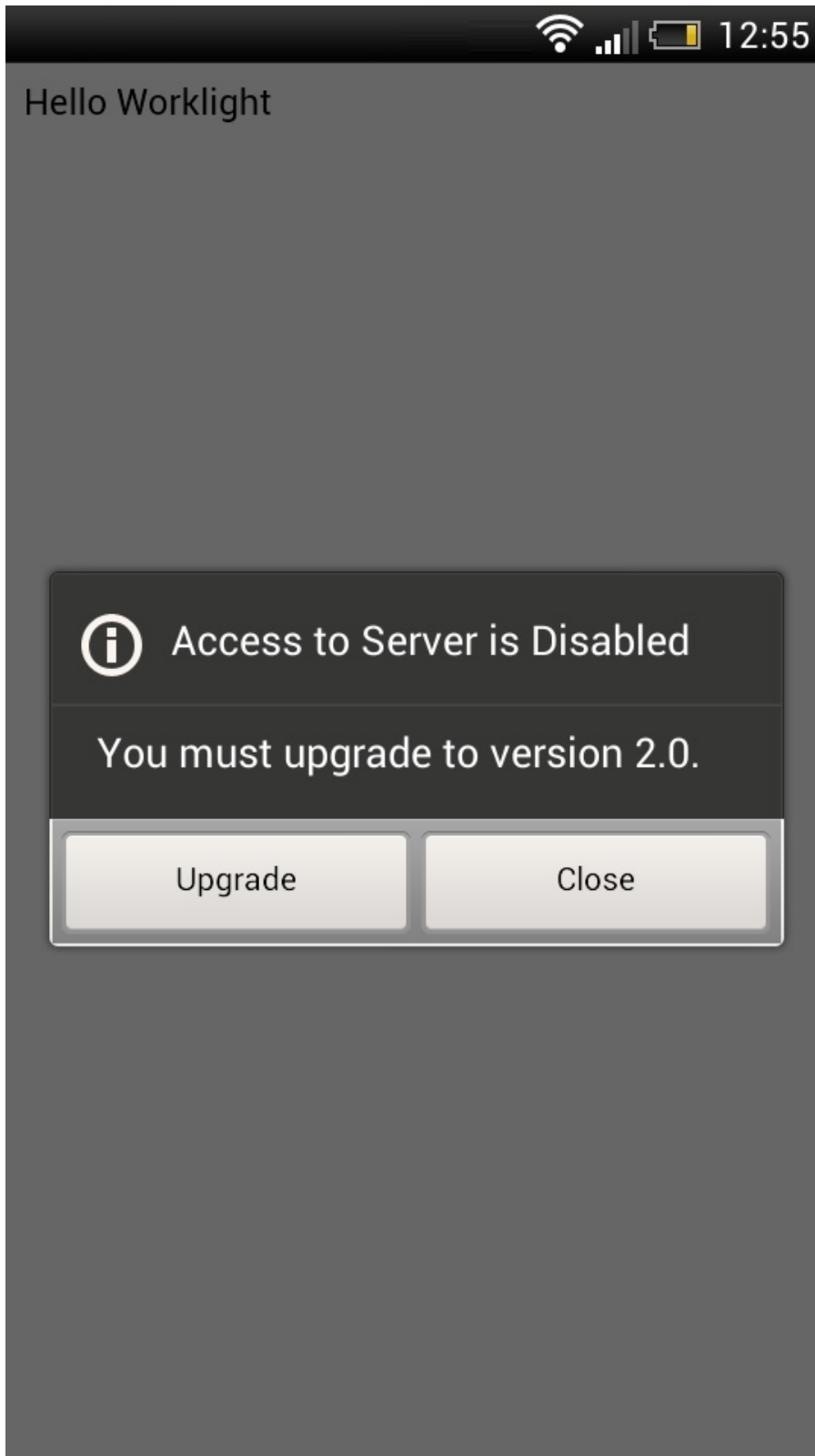
Figure 169. Remotely disabling an old version of an application

6. Click **Upgrade** to open the Application Center client. When the login details are correctly completed, you access the Details page of HelloWorld version 2.0

directly.
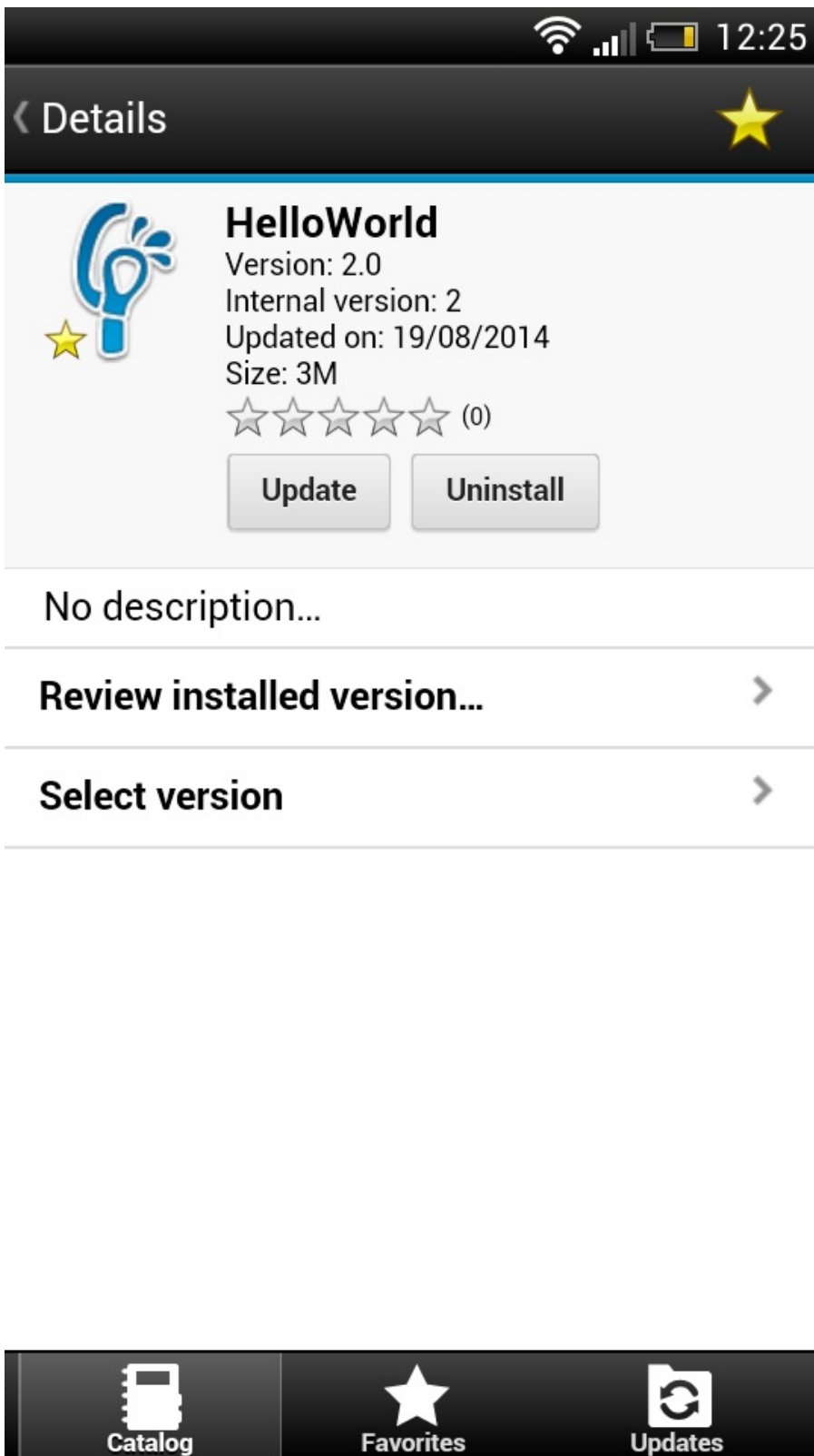
Figure 170. Details of HelloWorld 2.0 in the Application Center client

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**.

Figure 171. 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

Figure 172. 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 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.
- When you unregister a user, the reviews given by this user are not removed, but the name of the user is cleared.

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.

Figure 173. Local user groups

To delete a group, click the trash icon next to the group name. The group is also removed from the access control lists.

To add or remove members of a group, click the **Edit members** link of the group.
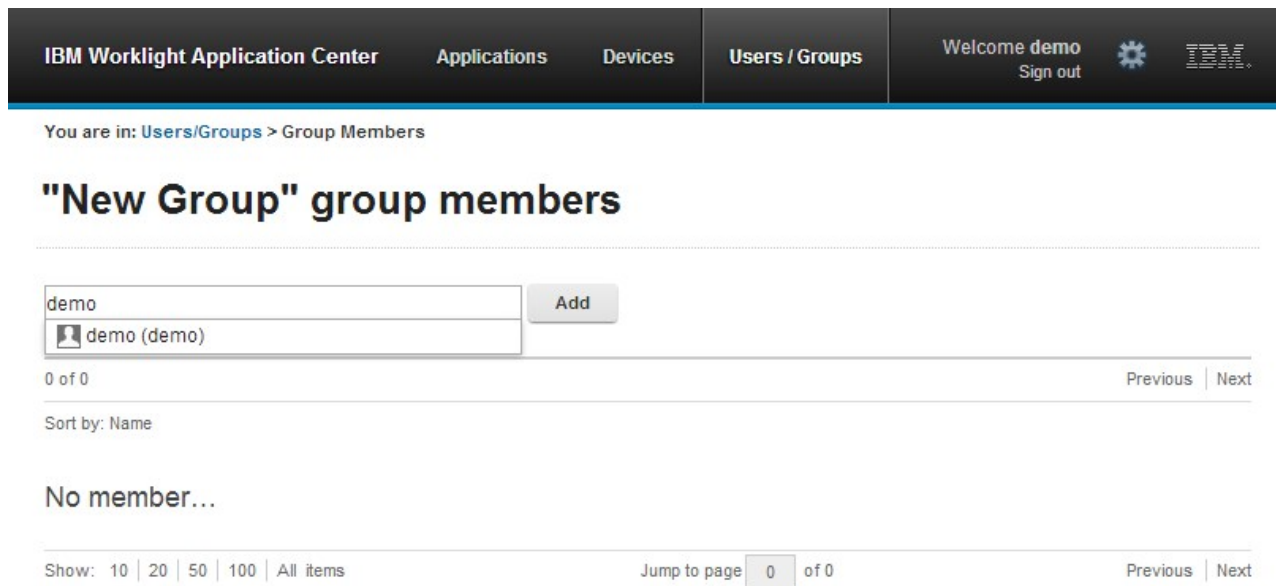
Figure 174. Managing group membership

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.

Figure 175. 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.

Figure 176. 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.

Figure 177. 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 178. 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*`/applicationcenter/installers.html`, where *hostname* is the host name of the server hosting the Application Center 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 Foundation" on page 13.

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
<?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 Worklight Studio IDE. In Worklight Studio, you can deploy Android application package (`.apk`) files, iOS application (`.ipa`) files, and BlackBerry (`.zip`) files that you choose from your file system. You can right-click an application (`.apk`, `.ipa`, or `.zip`) file to deploy it to the Application Center.

Procedure

To publish an application to the Application Center, complete the following steps:

1. Specify the publish preferences for the Application Center:
   a. In the main menu, click **Window** > **Preferences**.
   b. Expand **IBM Application Center** > **Publish Preferences**.
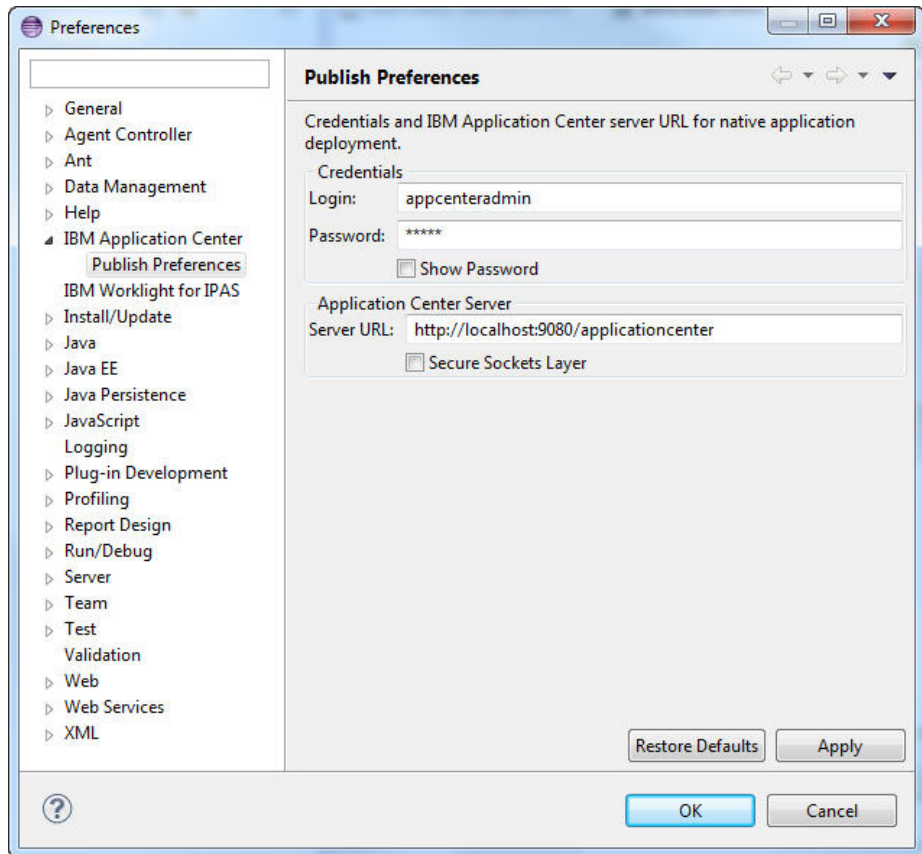
c. Specify the default publish preference settings for the Application Center:

Preference	Description
Credentials	Specify the login and password required to access the application repository.
Application Center Server	Specify the URL of the application center server to use when publishing applications.

2. Publish an application (`.apk`, `.ipa`, or `.zip` file) from a Worklight project:

   a. Right-click the application and click **IBM Application Center** > **Publish on IBM Application Center**. The Publish Confirm dialog opens.
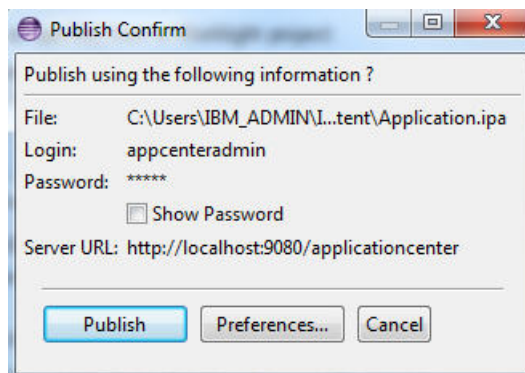
   b. In the Publish Confirm dialog, choose one of the following options:

Option	Description
Publish the application by using the current preferences.	Click **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.
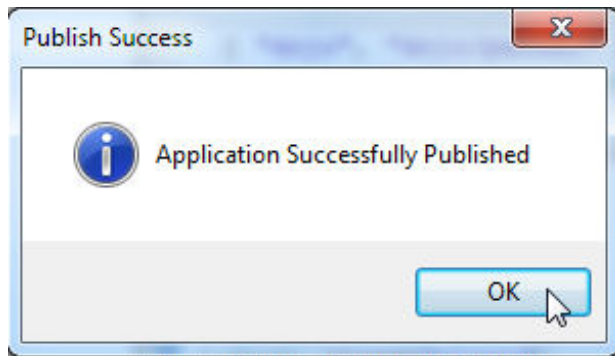
If the application already exists, publication will fail. You are given the option to overwrite the existing version of the application.
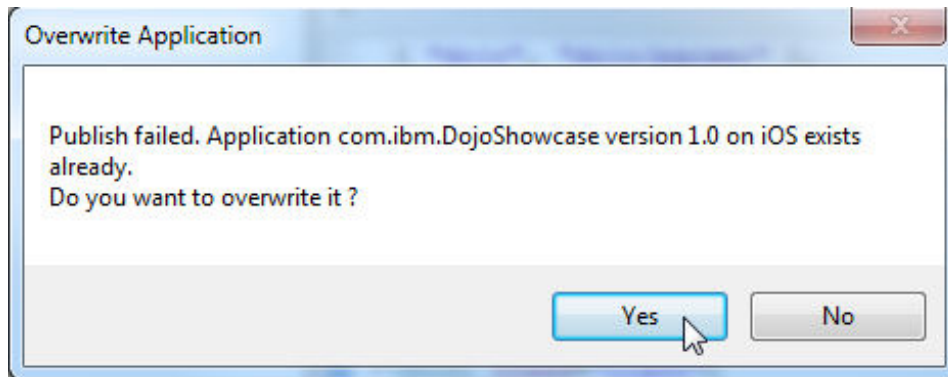
Tip: To publish an application that is not part of the Worklight project:

1) Right-click the Worklight project and click **IBM Application Center** > **Publish on IBM Application Center**. The Select Application to Publish window opens.
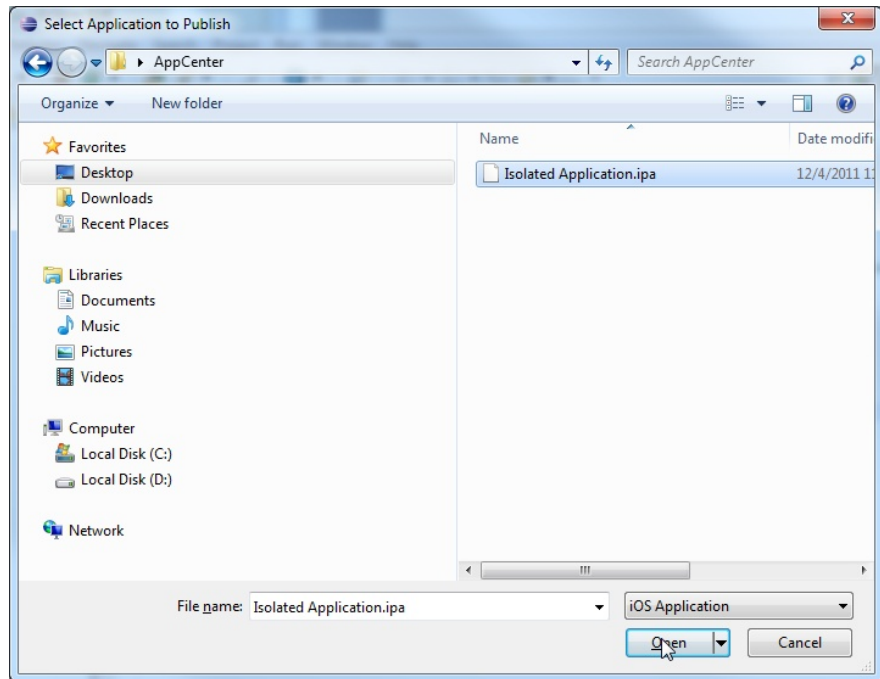
2) Navigate to the application (`.apk`, `.ipa`, or `.zip`) file that you want to publish and click **Open** to open the Publish Confirm dialog.

The mobile client

You can install applications on your mobile device with the Application Center mobile client.

The Application Center mobile client is the application that runs on your Android, iOS, 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/applicationcenter/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/applicationcenter/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 1350.

   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 1322.
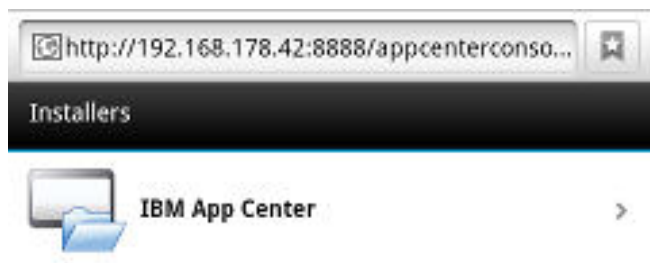
Figure 179. List of available mobile client applications to install

4. Select an item in the list to display the application details.

   Typically, these details include the application name and its version number.
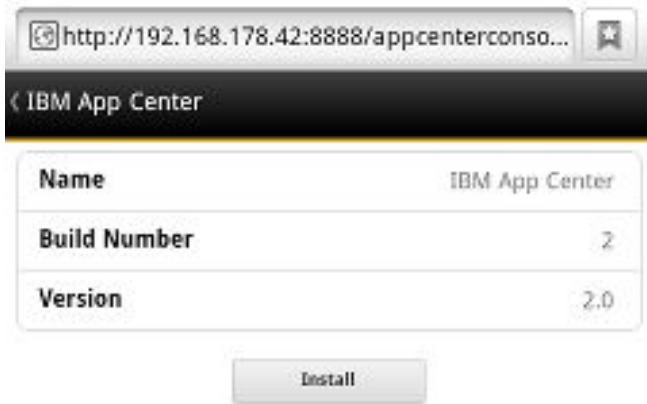
Figure 180. Application details

5. Tap **Install Now** to download the mobile client.
6. Launch the **Android Download** applications.
7. Select the Application Center client installer.

   You can see the access granted to the application when you choose to install it.
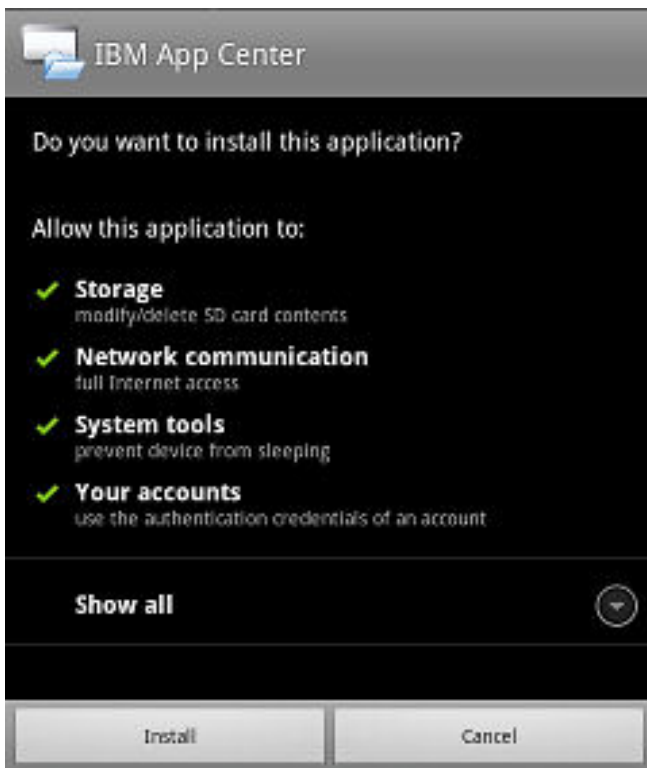
Figure 181. Installation of the mobile client on Android

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

Important: To install applications on iOS devices, you must first configure the Application Center server with SSL. See "Configuring Secure Sockets Layer (SSL)" on page 295.

`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 290.

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/applicationcenter/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/applicationcenter/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**.
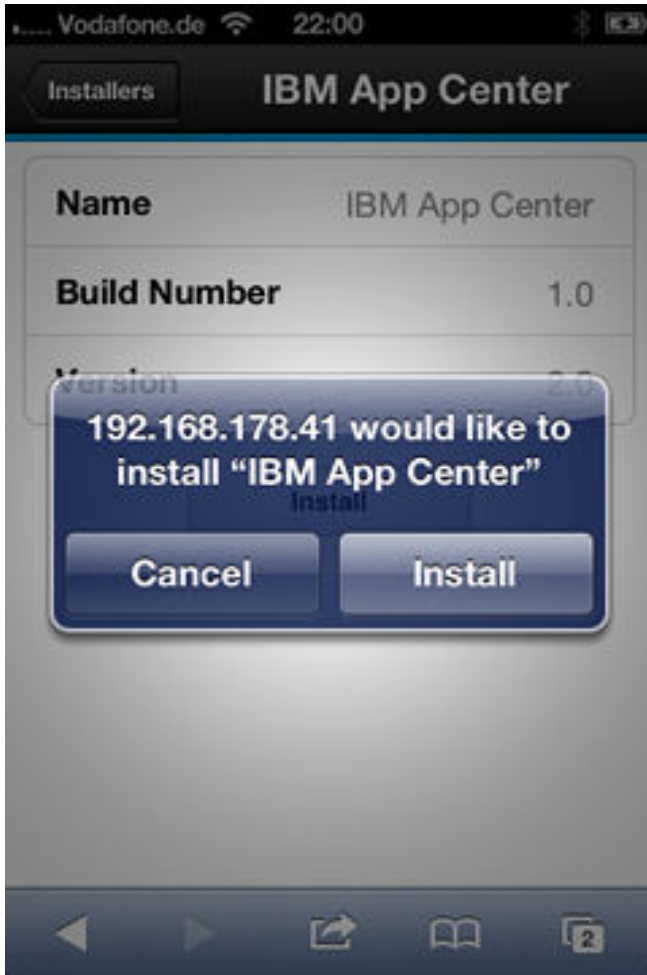
Figure 182. Confirm app to be installed

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/applicationcenter/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/applicationcenter/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.
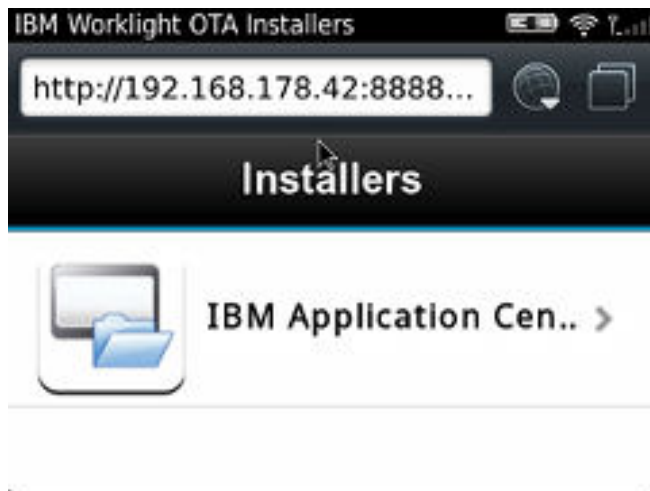
Figure 183. The installer in the BlackBerry browser

> **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 1340 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/applicationcenter/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/applicationcenter/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.
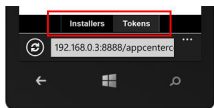
Figure 184. Preparing to install tokens and applications on a Windows Phone device

4. Tap **Tokens** and select an application enrollment token in the list of available tokens to display the token details.
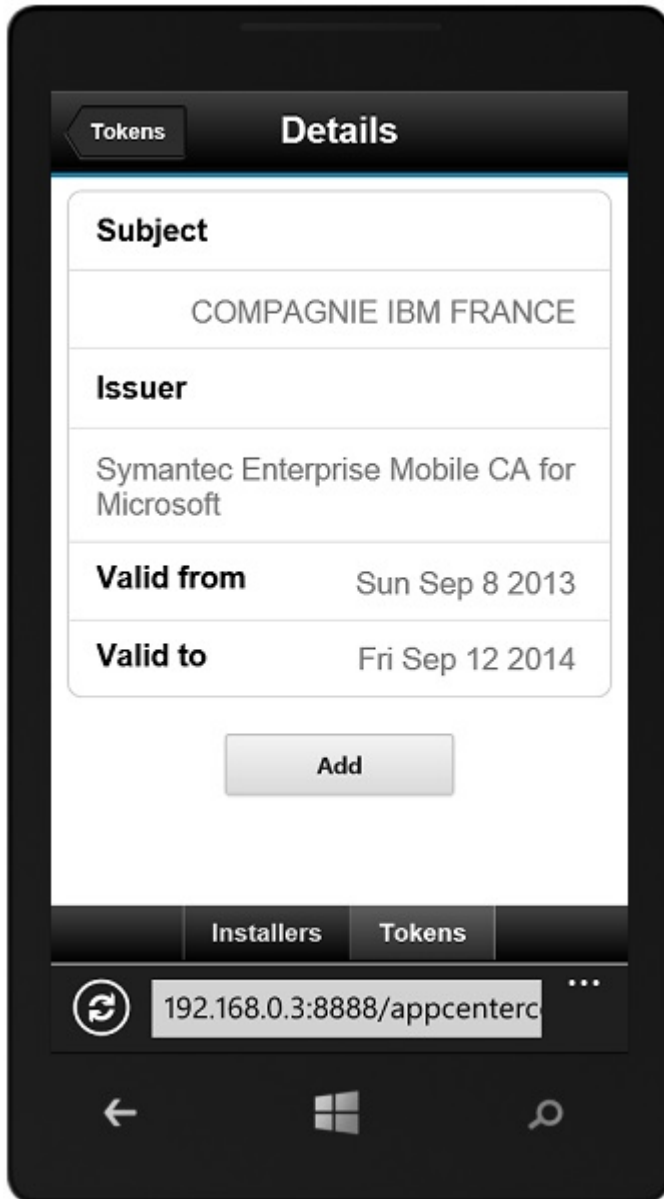
Figure 185. AET details on a Windows Phone device

    5. Tap **Add** to download the application enrollment token.

    6. Tap **Add** to add the company account.

Figure 186. 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.
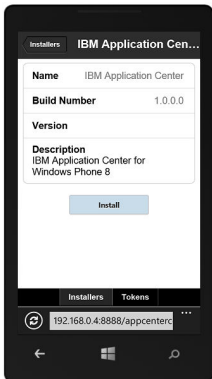
Figure 187. The application selected to download on a Windows Phone device

10. Tap **Install** to install the application.
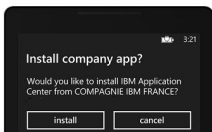
Figure 188. Installing the downloaded application on a Windows Phone device

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 1367 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 189. Login view on Android or iOS phone

Figure 190. Login view on Android or iOS tablet

Figure 191. 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 1299 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 192. Views in the client application (Android and iOS operating systems)
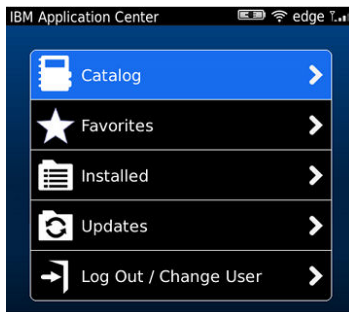
Figure 193. Views in the client application (BlackBerry devices)

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.
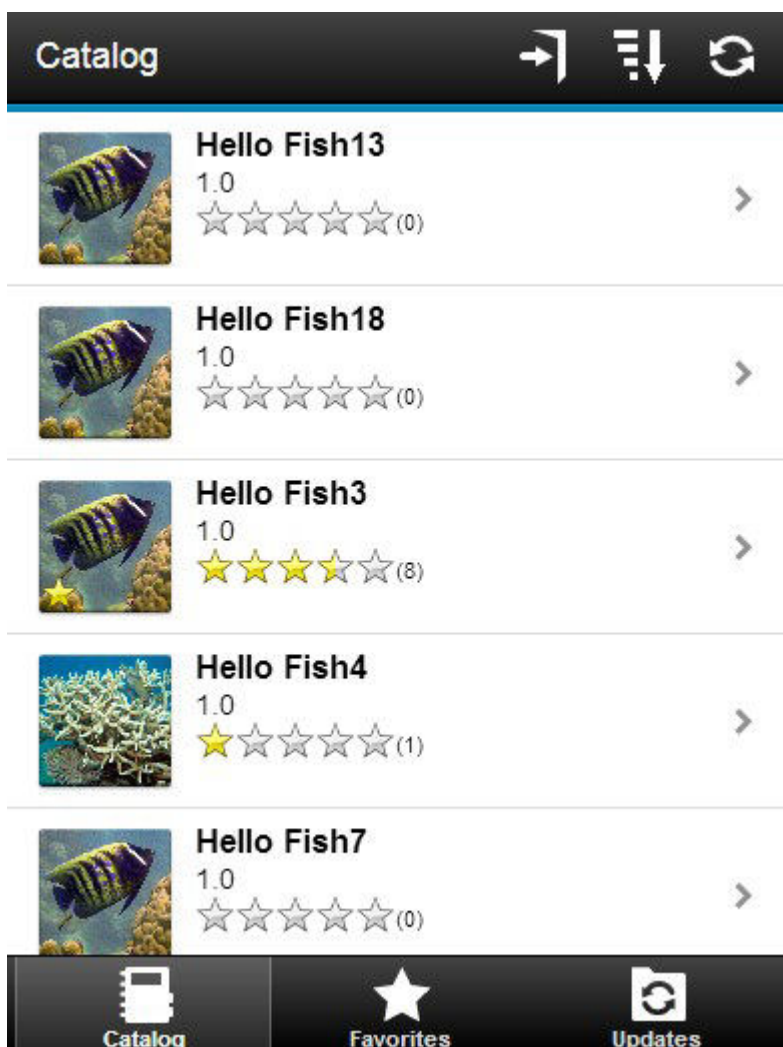
Figure 194. Catalog view on a phone
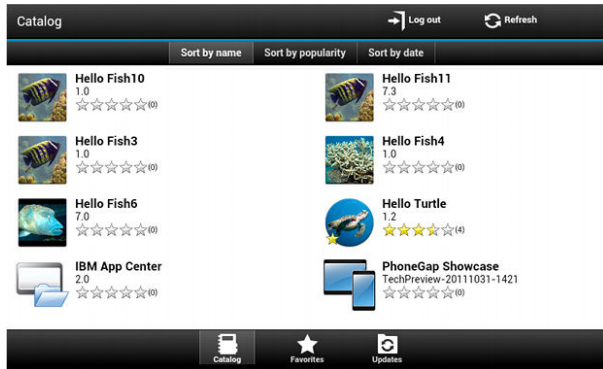
Figure 195. Catalog view on a tablet

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 1299 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 1322 for technical details concerning this
  property on all operating systems.
* Update date.
* Approximate size of the application file.
* Rating of the version and number of ratings received.
* Description of the application.

You can perform several actions in this view.

* Install, upgrade, downgrade, or uninstall an application version.
* Cancel the current operation in progress.
* Rate the application version if it is installed on the device.
* List the reviews of the this version or of all versions of the application.
* Show details of a previous version.
* Mark or unmark the application as a favorite app.
* Refresh the view with the latest changes from the Application Center server.

Installing an application on an Android device

From the **Details** view, you can install an application on your Android device.

About this task

In the **Details** view, if a previous version of the application is not installed, you
can install this application version on your Android device.
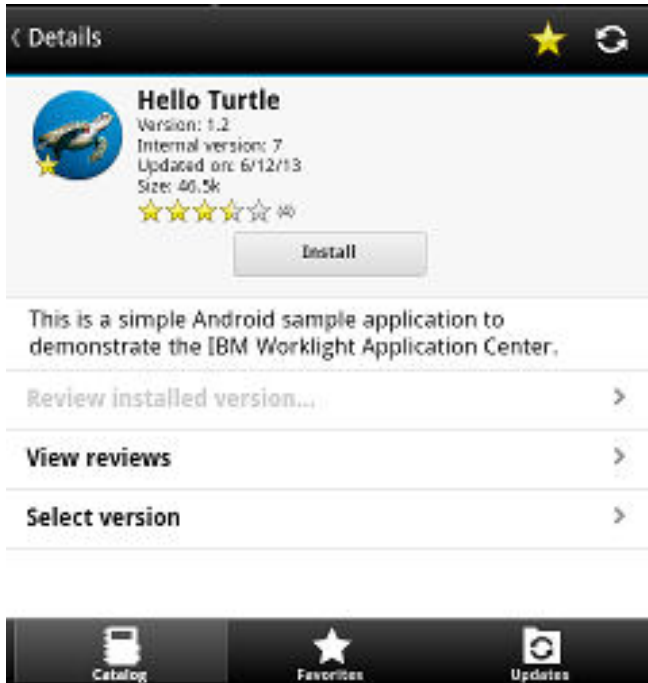
Figure 196. Details view of an app version shown on your Android device

Procedure

1. In the **Details** view, tap **Install**.

   The application is downloaded. You can tap **Cancel** in the **Details** view at any time during the download to cancel the download. (The **Cancel** button appears only during the installation steps.) If you let the download complete, you will see the rights that are granted to the application.

Figure 197. 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. When the application is successfully installed, it is also marked as a favorite app.

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 an application version on your iOS mobile device.

About this task
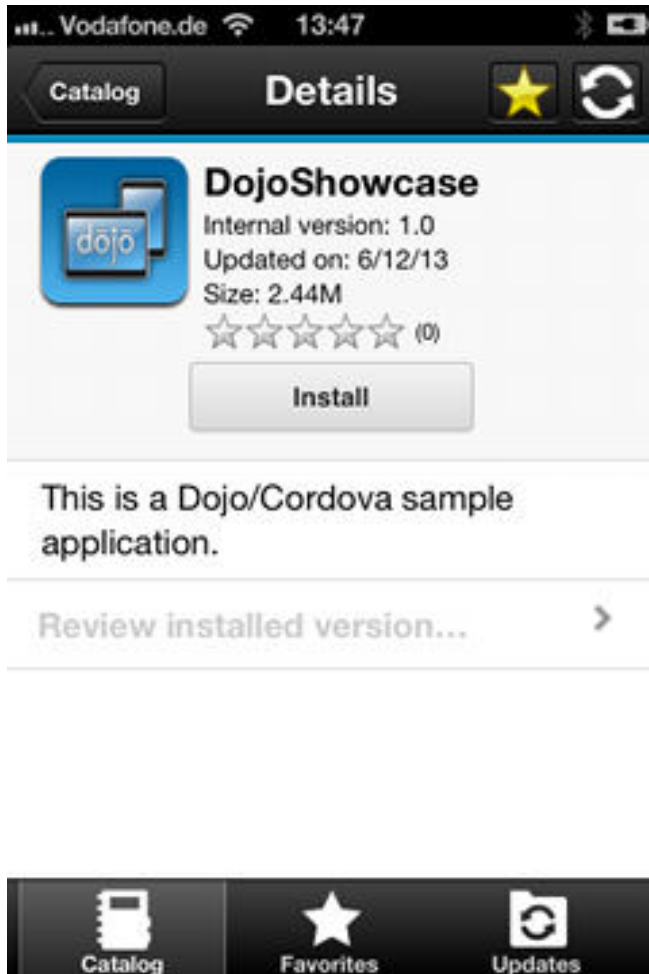
Figure 198. Details view of an app version shown on your iOS device

Important: To install applications on iOS devices, you must first configure the Application Center server with SSL. See "Configuring Secure Sockets Layer (SSL)" on page 295.

Procedure

1. In the **Details** view, tap **Install**. 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.
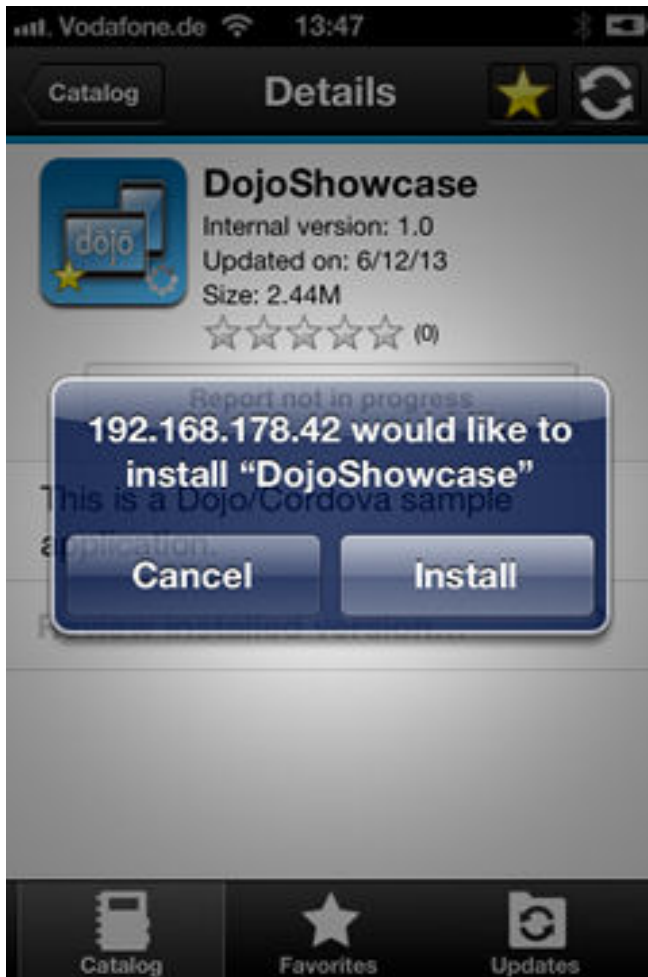
Figure 199. Canceling application installation on your iOS device

Depending on the action taken, the application is installed or not. When the application is successfully installed, it is also marked as a favorite app.

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.

Unlike the Android client, after the installation is finished, the **Install** button in the Details view does not change its label to **Uninstall**. In iOS, there is no **Uninstall** button. It is only possible to uninstall applications through the home screen.

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 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.
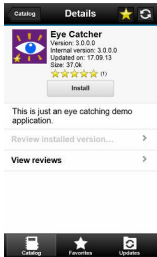
Figure 200. Details view of a version of a company application for installation on a Windows Phone device

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.

   The application is marked as a favorite app.
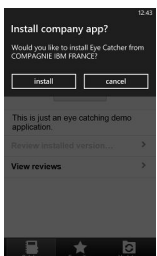
Figure 201. Confirming or canceling installation of a company application on a Windows Phone device

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. When the application is successfully installed, it is also marked as a favorite app.
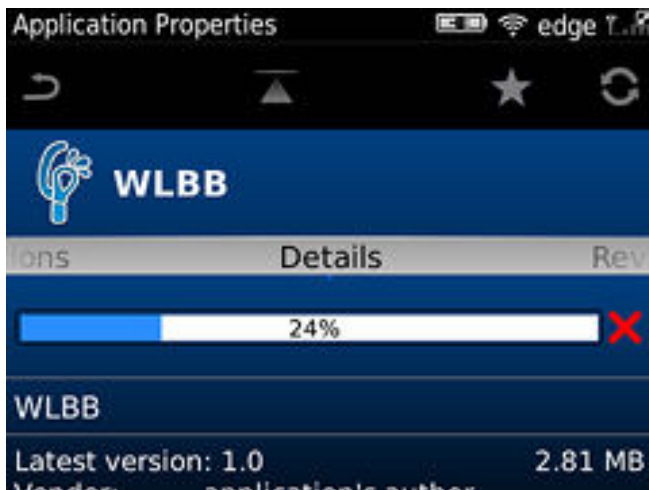
Figure 202. Downloading an application to a BlackBerry device

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 1319. 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.
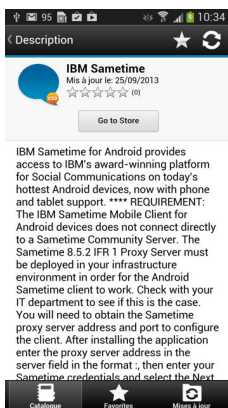
Figure 203. Accessing an application in Google play from the mobile client on the device
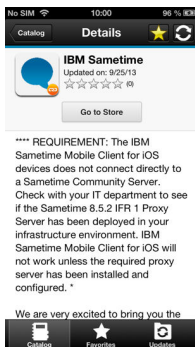
Figure 204. Accessing an application in Apple iTunes from the mobile client on the device

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 1372.
   - A tablet; see step 4 on page 1372.
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.
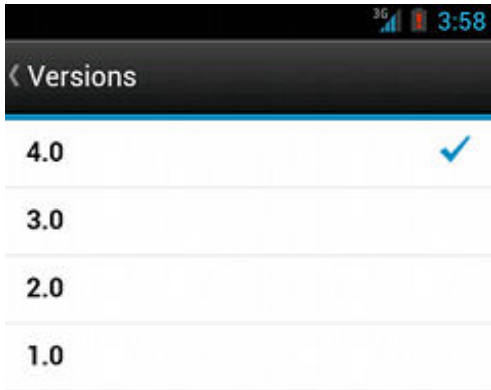
Figure 205. Specific version of an application selected in the list of versions on a Windows Phone, Android, or iOS phone

      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 206. 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 1364
   - "Installing an application on an iOS device" on page 1366
   - "Installing an application on a Windows Phone device" on page 1367
   - "Installing an application on a BlackBerry device" on page 1369

Upgrading the Application Center client automatically

You can enable automatic detection of new versions of the client application. Then, you can choose whether to download and install the new version on your mobile device.

Before you begin

Start the Application Center client.

About this task

New versions of the mobile client application that are available on the Application Center server can be detected automatically. When this feature is enabled, a more recent version of the application, if it exists, can be detected at start up or each time that the Available applications view is refreshed.

If a later version of the application is detected, you are requested to download and install the later version.

Automatic upgrade of the Application Center client application is enabled by default with the **appCenterAutoUpgrade** property set to `true`. This property is located in the Worklight project for the Application Center: `IBMAppCenter/apps/AppCenter/common/js/appcenter/config.json`.

If you want to disable automatic upgrade, you must set this property to `false` and rebuild the project for the required platforms.

Procedure

1. When a later version of the client is detected, tap **OK** to start the download and installation sequence.
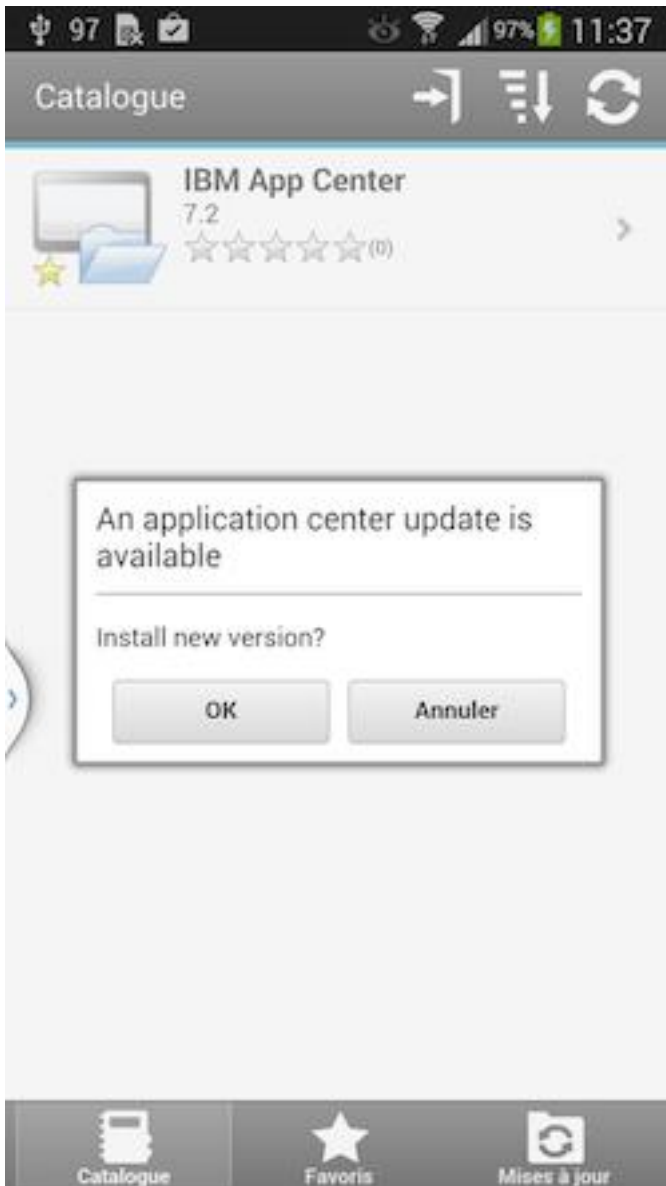
Figure 207. Detection of a later version of the client application available on the server

2. Tap **Install** to install the later version of the application.
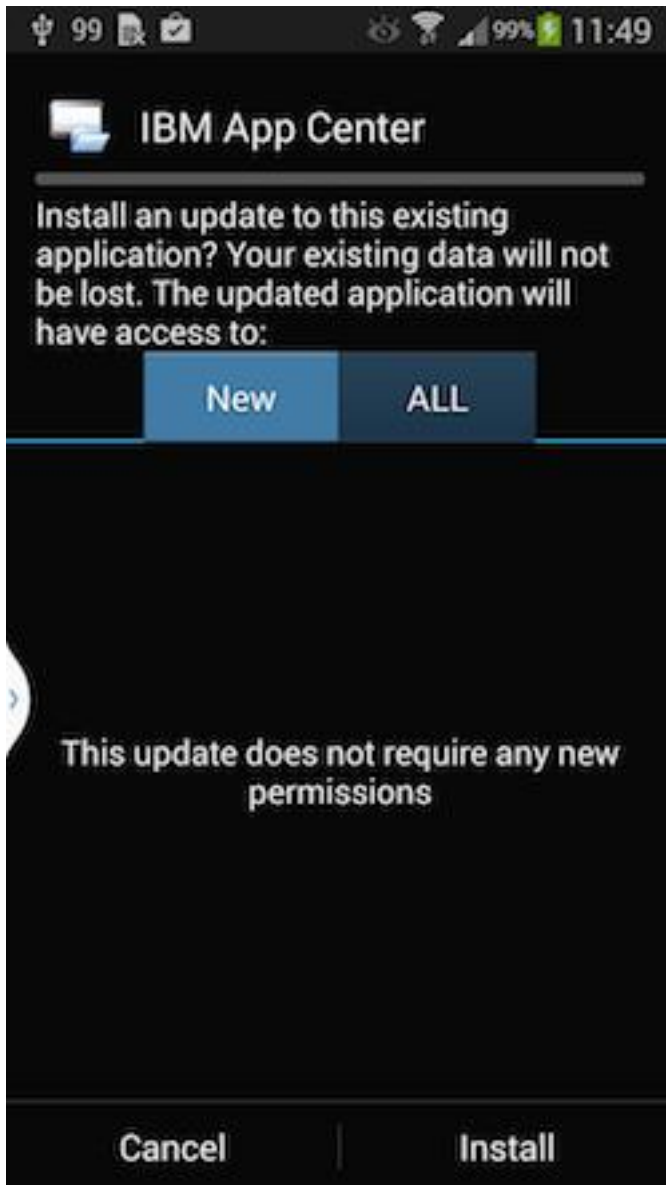
Figure 208. Confirm installation of the updated version of the application

3. Tap **Open** to start the updated application.
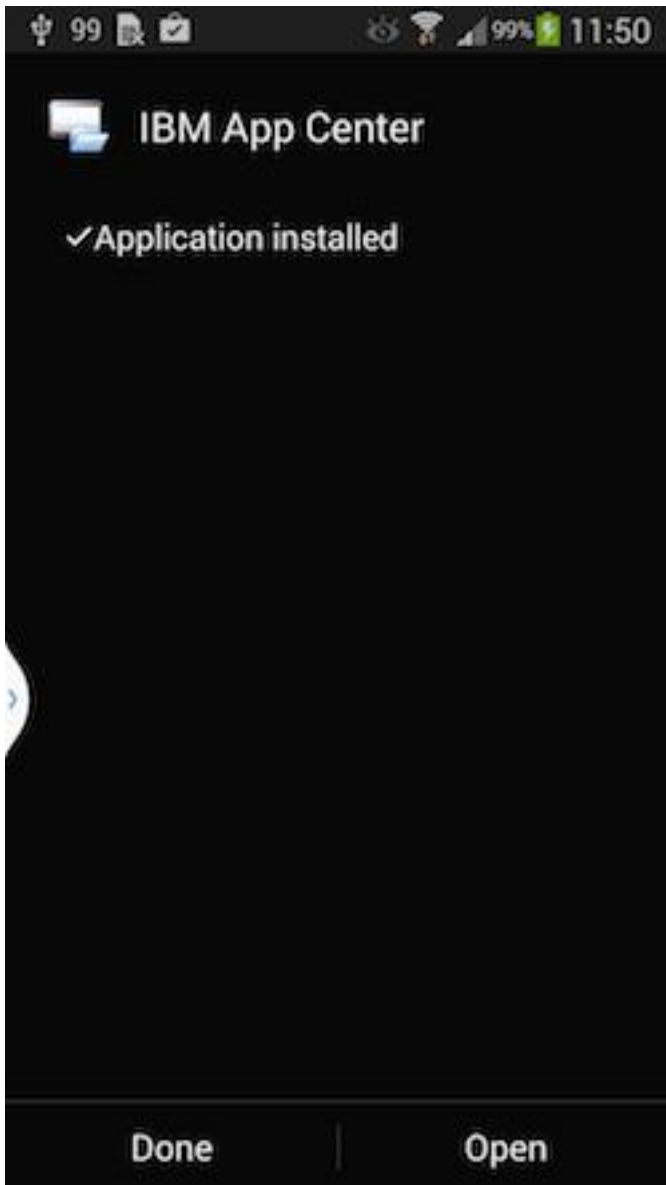
Figure 209. Starting the updated application

Results

You must log in to the updated version of the application to run it.

Figure 210. Logging in to the new version of the client application

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 1371 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 1299 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 1364.

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 1366.

On Windows Phone

Tap **Revert**. Follow the procedue documented in "Installing an application on a Windows Phone device" on page 1367.

On BlackBerry

Tap or click **Revert**. Follow the procedure documented in "Installing an application on a BlackBerry device" on page 1369.

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 ![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 314. 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 314. 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 314. 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 Foundation

IBM Worklight Foundation supports Federal Desktop Core Configuration (FDCC), and United States Government Configuration Baseline (USGCB) specifications. IBM Worklight Foundation also supports the Federal Information Processing Standards (FIPS) 140-2, which is a security standard that is used to accredit cryptographic modules.

For more information about the Federal Desktop Core Configuration and United States Government Configuration Baseline, see FDCC and USGCB.

For more information about the Federal Information Processing Standards 140-2, see FIPS 140-2 support.

FDCC and USGCB support

The United States federal government mandates that federal agency desktops that run on Microsoft Windows platforms adopt Federal Desktop Core Configuration (FDCC) or the newer United States Government Configuration Baseline (USGCB) security settings.

IBM Worklight V5.0.6 was tested by using the USGCB and FDCC security settings via a self-certification process. Testing includes a reasonable level of testing to ensure that installation and core features function on this configuration.

References

For more information about the Federal Desktop Core Configuration, see FDCC.

For more information about the United States Government Configuration Baseline, see USGCB.

FIPS 140-2 support

Federal Information Processing Standards (FIPS) are standards and guidelines that are issued by the United States National Institute of Standards and Technology (NIST) for federal government computer systems. FIPS Publication 140-2 is a security standard that is used to accredit cryptographic modules.

FIPS 140-2 on the Worklight Server, and SSL communications with the Worklight server

The IBM Worklight Foundation 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 Foundation 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 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 1385 for links to documentation to enable FIPS 140-2 mode in WebSphere Application Server.

FIPS 140-2 on the 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 Foundation. 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. FIPS is not yet supported on 64-bit architecture even though Worklight library does support 64-bit architecture. Therefore, FIPS must not be enabled on 64-bit target platform when XCode Build Setting (Architecture) is also set to 64 bit.
- 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 Worklight client can only communicate with a Worklight Server that runs in supported environments, which are listed in the System Requirements. If the Worklight Server runs in a non-supported environment, the HTTPS connection might fail with a "key size too small" error. This error does not occur with HTTP communications.
- 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 occurs over a non-FIPS 140-2 protected channel. If you want to use only a FIPS 140-2 protected channel, you must disable the Direct Update feature. To disable the Direct Update feature, you can override the mobile security test in the authenticationConfig.xml file in your server/conf directory.

```
<mobileSecurityTest name="mobileTests">
  <testDirectUpdate mode="disabled" />
</mobileSecurityTest>
```

Use this mobile security test for Android and iOS environments as follows:

```
<android securityTest="mobileTests" version="1.0">
```

or

```
<iphone bundleId="com.TestHybridApp" version="1.0" securityTest="mobileTests">
```

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 Foundation 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 211. Example
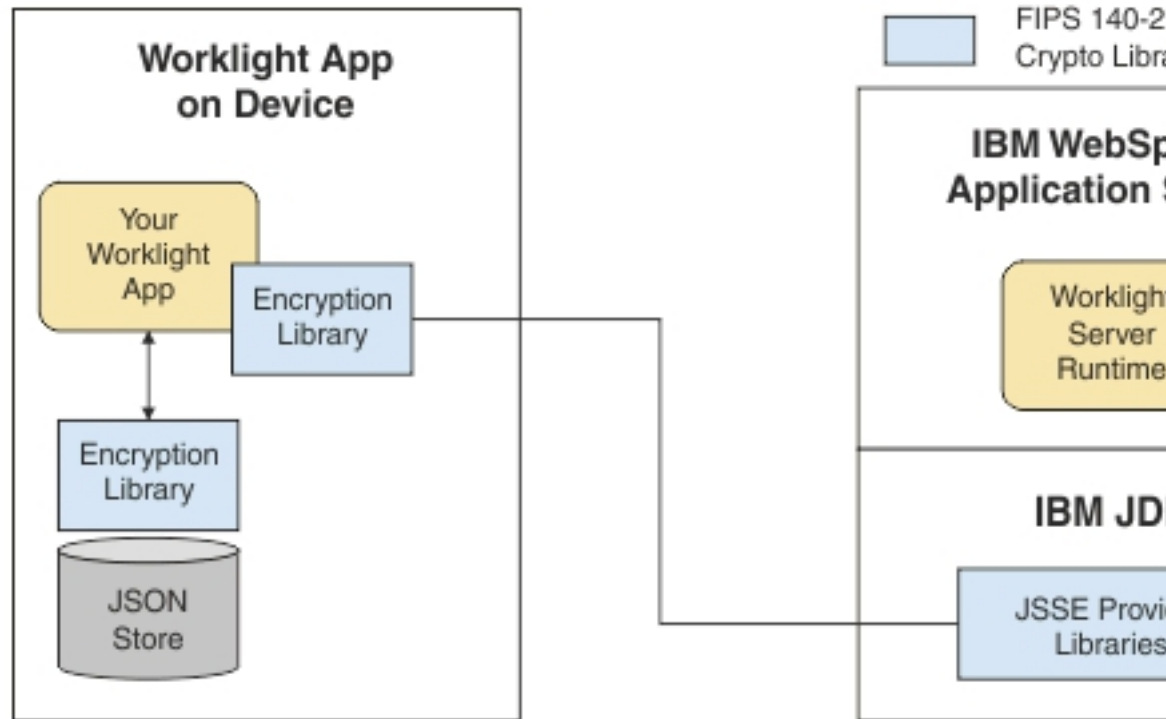
For more information about JSONStore, see "JSONStore overview" on page 787.

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.

Note: On Android, FIPS 140-2 is not supported on 64-bit architecture even though the Worklight library does support 64-bit architecture. When you use FIPS 140-2 on a 64-bit device, you might see the following error:

```
java.lang.UnsatisfiedLinkError: dlopen failed: "..." is 32-bit instead of 64-bit
```

This error means that you have 64-bit native libraries in your Android project, and FIPS 140-2 does not currently work when you use these libraries. To confirm, go to `src/main/libs` or `src/main/jniLibs` under your Android project, and check whether you have the `x86_64` or `arm64-v8a` folders. If you do, delete these folders, and FIPS 140-2 can work again.

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**.
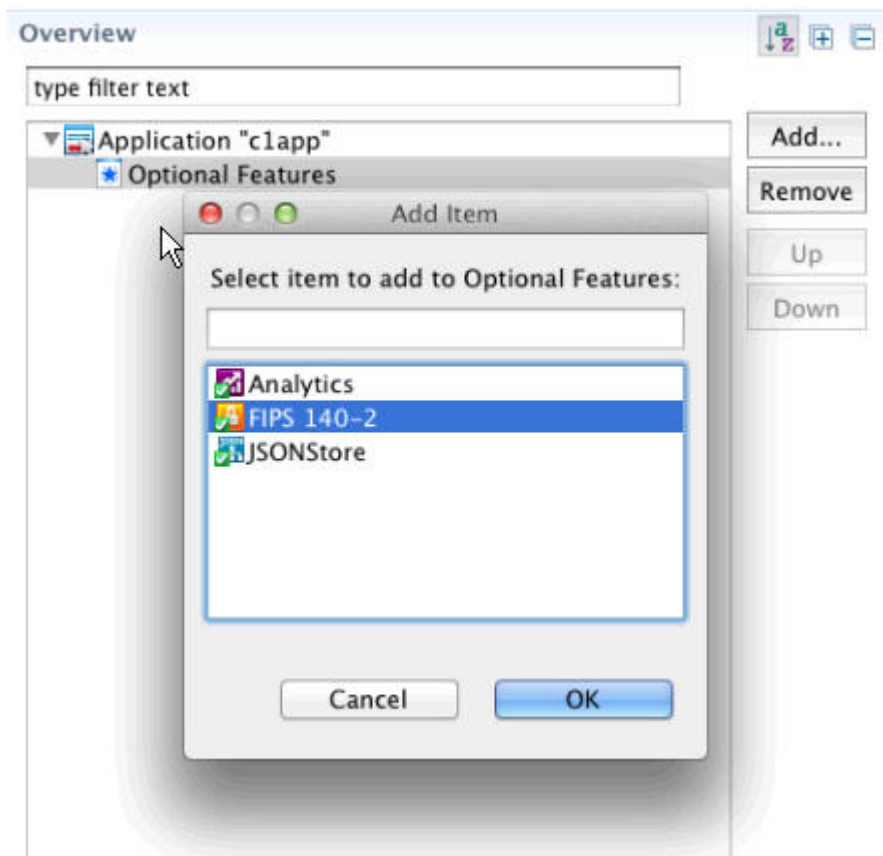
Figure 212. Installing the FIPS 140-2 optional feature

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 Foundation 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 Andro
  //   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 Foundation).

```
$(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 1385.

Configuring FIPS 140-2 for existing applications

You must modify applications that were created in earlier versions of IBM Worklight Foundation 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 1385. 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 1385, 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 Foundation 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.

Chapter 14. Monitoring and mobile operations

IBM Worklight Foundation includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM Worklight Foundation applications and servers, and for monitoring server health.

Logging and monitoring mechanisms

IBM Worklight Foundation 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 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 Worklight logs. 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 315. 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://ibm.biz/knowctr#SSEQTP_7.0.0/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.0	http://ibm.biz/knowctr#SSEQTP_8.0.0/com.ibm.websphere.base.doc/info/aes/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 full profile	http://ibm.biz/knowctr#SSEQTP_8.5.5/com.ibm.websphere.base.doc/ae/ttrb_trcover.html
WebSphere Application Server Version 8.5 Liberty profile	http://ibm.biz/knowctr#SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/rwlp_logging.html?cp=SSEQTP_8.5.5%2F1-16-0-0

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 IBM Operations Analytics - Log Analysis or other industry standard log file monitoring tools to monitor logs and highlight errors and warnings.

For WebSphere Application Server, use the log viewing facilities that are described in the IBM Knowledge Center at the URLs 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 log for administration operations

Worklight Console stores an audit log for login, logout, and for all administration operations, such as deploying apps or adapters or locking apps. The audit log can be disabled by setting the JNDI property **ibm.worklight.admin.audit** on the web application of the Worklight Administration service (`worklightadmin.war`) to false.

When the audit log is enabled, you can download it from Worklight Console by clicking the **Audit log** link in the footer of the page.

Audit logs for adapters

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 Worklight vitality queries to run a health check of your server, and determine the vitality status of your server.

You generally use the 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 316. Examples of queries that help determine server vitality

Query	Purpose

`http://<server>:<port>/`
`<publicWorkLightContext>/ws/rest/`
`vitality`
	Checks the server as a whole.

Table 316. Examples of queries that help determine server vitality (continued)

Query	Purpose
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 three 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>
  <ALERT>
    <DATE>2014-07-08T11:39:42.622+0300</DATE>
    <EVENTID>0</EVENTID>
    <SYSTEM>WRKL</SYSTEM>
    <SUBJECT>BUILD</SUBJECT>
    <COMPUTER>192.168.218.1</COMPUTER>
    <DESCRIPTION>6.2.0.00.20140707-1736</DESCRIPTION>
  </ALERT>
</ROOT>
```

Return values

The following table lists the attributes that might be returned, and their possible values.

Table 317. Return values and values

Return attribute	Possible values
DATE	Date value in JavaScript™ format

Table 317. Return values and values (continued)

Return attribute	Possible values
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
	BUILD for the version of the Worklight Server
TYPE	I – valid
	E – error
COMPUTER	Reporting computer name
DESCRIPTION	Status description in plain text

The returning XML contains more attributes, which are undocumented constants that you must not use.

Configuring logging in the development server

Information about the default logging settings for the embedded development, and procedures for changing them.

When you are trying to diagnose problems in the Worklight Studio embedded test server (for example, when debugging a custom login module), it is important to be able to see log messages. The default settings for server logging are described in this section, along with the procedures for changing them if you must see finer levels of messages.

In previous releases of Worklight Studio, the embedded Jetty test server did not allow viewing the server logs. In Worklight Studio V6.0.0, the test server was 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:
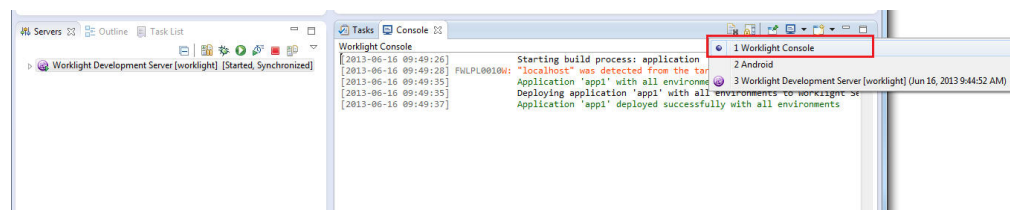
Changing the logging levels of a web application in Liberty profile

Changing the logging properties for individual application server types is done with those servers' administration tools.

To provide two examples for WebSphere Application Server Liberty profile, the `server.xml` file can be modified by appending the new `logging` element:
- To enable the `INFO` logging level for the server console, the following line is added to the `server.xml` file:

  `<logging consoleLogLevel="INFO"/>`
- To enable trace log files, the following line is added to the `server.xml` file:

  `<logging traceSpecification="*=audit=enabled:com.worklight.*=info=enabled" />`

The available setting levels for `consoleLogLevel` are:
- `INFO`
- `AUDIT`
- `WARNING`
- `ERROR`
- `OFF`

Please note that this parameter does not support `DEBUG` level logging.

No server restart is necessary after you modify these settings.

Whatever the logging level, the messages are displayed in Worklight Studio in its console view with the name **Worklight Development Server**, as shown in the following screen capture:
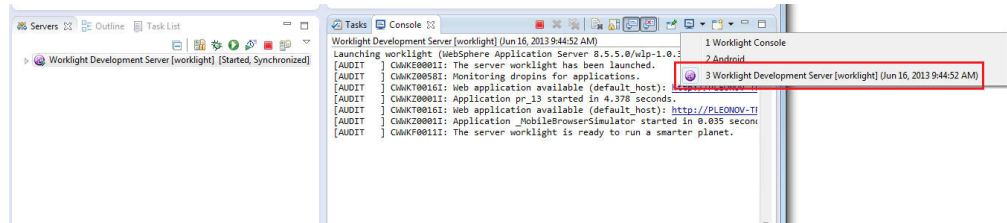
This console view allows you to see messages from the Worklight Development Server, but with some known limitations:

- Localized log messages are shown incorrectly. For more information about this issue, see Liberty profile: Trace and logging.
- Setting the value of `consoleLogLevel` to `WARNING`, `ERROR`, or `OFF` causes the server not to start from Worklight Studio using the Eclipse `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.

IBM Worklight Foundation includes a scalable operational analytics feature that is accessible from the Worklight Console. The analytics feature enables enterprises to search across logs and events that are collected from devices, apps, and servers for patterns, problems, and platform usage statistics.

The data for operational analytics includes the following sources:
- Interactions of any app-to-server activity (anything that is supported by the Worklight client/server protocol, including push notification).
- Client-side logs and crashes.
- Server-side logs that are captured in traditional Worklight log files.

The operational analytics feature is accessible from the Worklight Console and includes these capabilities:

Interactive web-based usage.

> **Dashboard view**
>> These features include interaction support to see the full device usage across the platform for the last 30, 60 or 90 days. You can drill down to specific apps and app versions.
>
> **Devices view**
>> View device information, including session activity, network activity, and JSONStore analytics. Search is provided to view information about a particular device.
>
> **Adapters view**
>> View information about adapters such as invocation frequency and network latency. You can drill down to a specific adapter or procedure.
>
> **Servers view**
>> View analytics information about individual servers in a cluster. Download server logs.
>
> **Activities view**
>> View analytics data on custom activities that are created with the client-side logging features.
>
> **Search view**
>> Provides a text search for client and server-side logs. Allows filtering by application, version, environment, severity, and so on.

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 Analytics Platform, enabling the BIRT feature is redundant. 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 1423 feature is a subset of the total data that is collected as part of the Operational "Analytics" on page 1395 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 318. Comparison of analytics and reports features. This table lists a 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 the IBM Worklight Analytics Platform	Relational database
Analytics mechanism	Each log event is treated as a JSON document. The data in the document is indexed so that it can be searched by keyword in the document and presented in a canonical form that shows the app, version, some device data, location (if enabled), timestamp, adapter (if present in the document) and other data.	Each log event is treated as a row in the raw Reports database table and then aggregated for statistics into the `app_activities` database table, summarized to app, device operating system, and timestamp relationships.
Access mechanism	Worklight Console	BIRT or other reporting tools that can understand data cubes
Extendable	Extending the published reports is not supported.	Data can be extracted from the database tables by using any means that you desire, including but not limited to, BIRT.
Search across logs	Yes	No
Optional	Yes	Yes

In addition to an at-a-glance view of your mobile and web application analytics, the operational analytics includes the capability to perform raw search against server logs, client activities, captured client crash data. The operational analytics

feature can also search any additional data that you explicitly provide through client and server-side API function calls that feed into the IBM Worklight Analytics Platform.

Operational analytics

The operational analytics platform collects data about applications, adapters, devices, and logs to give a high-level view of the client interaction with the Worklight Server and to enable problem detection.

The data for operational analytics includes the following sources:
- Crash events of an application on iOS and Android devices (crash events for native code and JavaScript errors).
- Interactions of any application-to-server activity (anything that is supported by the Worklight client/server protocol, including push notification).
- Server-side logs that are captured in traditional Worklight log files.

The operational analytics feature is accessible from the Worklight Console and includes the following capabilities:
- Near real-time analytics for client activity with the Worklight Server.
- Analytics for adapter hits.
- Network latency analytics.
- Client log search and download.
- Server log search and download.
- Crash and stack trace search.

In addition to an at-a-glance view of your mobile and web application analytics, the analytics feature includes the capability to perform a raw search against server and client logs, captured client crash data, and any extra data you explicitly provide through client and server API function calls that feed into the IBM Worklight Analytics Platform.

Data capture
When the IBM Worklight Analytics Platform is deployed and the Worklight Server is properly configured, data begins to flow from the Worklight Server to the IBM Worklight Analytics Platform. Some types of data are captured automatically without extra client or server configurations. Some types of data require changes to be made in the client application to capture or forward the data to the Worklight Server.

Analytics event types

The following image shows the analytics data flow:
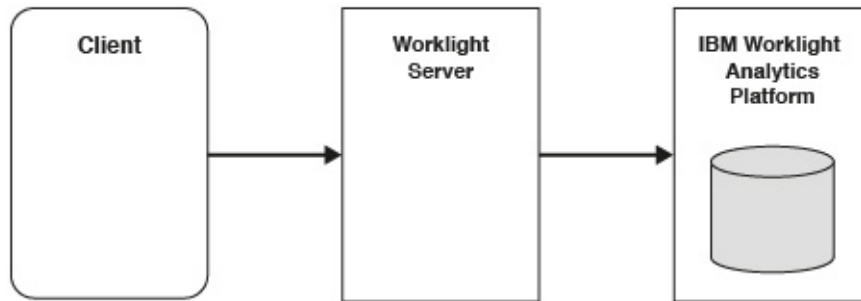
All data that is sent from the Worklight Server to the IBM Worklight Analytics Platform is categorized by its type. This section briefly describes the different types of analytics data that is captured and analyzed.

App Activities

> All IBM Worklight Foundation client/server network communication is considered to be an app activity. An app activity is sent to the IBM Worklight Analytics Platform when:
>
> - A client device begins a new session with the Worklight Server.
> - A client device makes an adapter request.
>
> When the client communicates with the Worklight Server through one of the previously mentioned events, it also sends metadata about the device, including:
>
> - Device environment (Android, iOS, Windows Phone, and so on).
> - Device model (iPhone, iPad, Nexus, and so on).
> - Device OS version (6.2, 4.2.2, and so on).
>
> Extra information that is captured during a client/network communication includes:
>
> - Response times for adapter calls.
> - Response payload sizes for adapter calls.

Server Logs

> Normal Worklight Server activity produces log messages that are saved to the disk. These messages are also forwarded to the IBM Worklight Analytics Platform and can be searched.

Client Logs

> Client devices can be configured to capture log data and crash events to be forwarded to the Worklight Server. For more information, see "Manually captured data" on page 1399.

Notification Activities

> Upon a successful push notification, a notification activity is automatically sent to the IBM Worklight Analytics Platform.

Note: No session data is sent to the IBM Worklight Analytics Platform Server until the application is connected to the Worklight Server. A connection to the Worklight Server can be achieved by calling `WL.Client.connect()`. Adapter calls can still be logged in the IBM Worklight Analytics Platform Server without a successful connection.

A session occurs when an `init` activity is sent to the server. An `init` activity occurs only when `WL.Client.connect()` is invoked.

Manually captured data

The following data must be captured manually by changing the client application.

Client Logs

The following examples show how to create client logs to be sent to the IBM Worklight Analytics Platform.

Hybrid applications (JavaScript):

```
// Set logging level (default level is FATAL).
WL.Logger.config({level: "DEBUG"});

// Log the message.
WL.Logger.debug("This message is persisted locally until it is sent to the server");

// Call the 'send' method explicitly to send the logs to the IBM Worklight Server.
WL.Logger.send();
```

Native iOS applications:

```
#import "OCLogger.h"

// Set logging level (default level is FATAL).
[OCLogger setLevel:OCLogger_DEBUG];

// Create a new instance of the logger and log the message.
OCLogger *logger = [OCLogger getInstanceWithPackage:@"MyPackage"];
[logger debug:@"This message is persisted locally until it is sent to the server"];

// Call the 'send' method explicitly to send the logs to the IBM Worklight Server
[OCLogger send];
```

Native Android applications:

```
import com.worklight.common.Logger;

// Set logging level (default level is FATAL).
Logger.setLevel(LEVEL.DEBUG);

// Create a new instance of the logger and log the message.
Logger logger = Logger.getInstance("MyPackage");
logger.debug("This message is persisted locally until it is sent to the server");

// Call the 'send' method explicitly to send the logs to the IBM Worklight Server
Logger.send();
```

For more information about capturing client-side logs, see "Client-side log capture configuration from the Worklight Console" on page 984.

All persisted client-side logs are sent automatically upon a successful initialization (successful session start) with the Worklight Server. An explicit API is provided if you want to send the logs more frequently.

Client Network Activities

Network activities for client devices are captured and persisted on the device automatically. However, they are only sent when the client successfully initializes with the Worklight Server (successful session start).

Analytics Logging

The client-logging feature enables developers to create logs to help debug problems and capture errors. A separate API exists for creating logs that are not meant for problem detection and capture:

```
WL.Analytics.log( object, message );
```

For example:

```
WL.Analytics.log( { "hello": "world" } , "This is an analytics log" );
```

Logs that are created by the client-side logger are only captured based on the logging level that is set. For example, DEBUG logs are not captured when the logging level is set the FATAL. However, logs that are produced by `WL.Analytics.log` are always captured, despite the current logging level.

The metadata object is not searchable in the analytics console. The primary purpose of the metadata object is to allow developers to log custom JSON objects if they wish to export analytics data into their own custom tool.

Crash reports

If you want uncaught exceptions recorded and sent to IBM Worklight Analytics Platform in your native iOS application, you can do so by registering an uncaught exception handler.

```
NSSetUncaughtExceptionHandler(&unCaughtExceptionHandler);
```

Then you can implement your uncaught exception handler, utilizing `OCLogger` to record information about the exception.

```
static void unCaughtExceptionHandler(NSException *e) {
  [[OCLogger getInstanceWithPackage:@"example.package"] fatal:@"Uncaught Exception: %@. Reas
}
```

Note that the logs will not get sent to the server until the application is restarted and connects to the Worklight Server.

Custom Activities

The operational analytics console provides a page to view analytics for custom activities. These activities can be created on the client-side by the `WL.Analytics` API. Using the `_activity` key in the object for the `WL.Analytics` call creates a new activity on the server. For example:

```
WL.Analytics.log( { "_activity" : "myCustomActivity" } );
```

creates a new activity in the IBM Worklight Analytics Platform that can be searched on the Activities page of the analytics console.

JSONStore Analytics

For more information about configuring the client to send analytics data for JSONStore, see "JSONStore analytics" on page 836.

Client configurations

No additional client configurations are needed for client devices to forward analytics data to the Worklight Server.

The Analytics Optional Feature is not required to be enabled on the client for analytics in IBM Worklight Foundation V6.2.

The `analyticsEnabled` flag in the `initOptions.js` file is not required to be enabled on the client for analytics in IBM Worklight Foundation V6.2.

These configurations are only necessary if you are using the previous analytics platform (IBM SmartCloud Analytics Embedded). If you are using the new IBM Worklight Analytics Platform in IBM Worklight Foundation V6.2, then these properties can be ignored.

IBM Worklight Analytics Platform security
Learn about security with the IBM Worklight Analytics Platform.

Protecting the analytics data entry point with basic authentication:

You can configure the security of the IBM Worklight Analytics Platform to protect the data entry point by using basic authentication.

About this task

Data is sent from the Worklight Server to the IBM Worklight Analytics Platform when the following IBM Worklight Foundation property is set:

`wl.analytics.url=http://<hostname>:<port>/<context-root>/data`

The IBM Worklight Analytics Platform exposes this path and it is unprotected by default. This path can be protected with basic authentication (user name and password) by using the following steps.

Procedure

1. Open the `WEB-INF/web.xml` file in the WAR file. You must extract the WAR file or use a tool to manipulate compressed files.
2. Search for a commented-out section that is called **Basic Authenication**, and uncomment the section.

```
<!-- Basic Authentication -->

  <security-constraint>
    <web-resource-collection>
      <web-resource-name>data</web-resource-name>
      <url-pattern>/data</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <role-name>admins</role-name>
    </auth-constraint>
  </security-constraint>

  <login-config>
    <auth-method>BASIC</auth-method>
  </login-config>
```

3. Replace the role name with the role that is configured for your server.
4. In the `worklight.properties` file, set the `wl.analytics.username` and `wl.analytics.password` values for basic authentication. For more information about encrypting sensitive information in the `worklight.properties` file, see "Storing properties in encrypted format" on page 1178.

```
wl.analytics.username=bob
wl.analytics.password=password
```

Results

After you complete these steps, the IBM Worklight Analytics Platform no longer accepts any incoming data unless the request also contains the correct credentials. When the `wl.analytics.username` and `wl.analytics.password` values are set, the Worklight Server uses these credentials when it forwards data to the IBM Worklight Analytics Platform. For more information about IBM Worklight Foundation properties, see "Worklight properties" on page 1419.

Note: These configuration steps protect only the `/data` path that accepts incoming data. It does not protect the console.

Example

The following example shows the equivalent configuration in the Liberty
`server.xml` file:

```
<application context-root="/analytics"
             id="analytics"
             location="analytics.war"
             name="analytics"
             type="war">
  <application-bnd>
    <security-role name="admins">
      <user name="bob" />
    </security-role>
  </application-bnd>
</application>

<basicRegistry id="basic">
  <user name="bob" password="password" />
</basicRegistry>
```

Ports that are used by the IBM Worklight Analytics Platform:

When the IBM Worklight Analytics Platform is started, it listens on a predefined
port. In V6.2.0.1 and earlier, it listens on ports 9500 and 9600. In the latest interim
fix, it listens on port 9600.

Port 9500 - HTTP Port
> This port can be used for HTTP requests that are made directly to the IBM
> Worklight Analytics Platform. It is not required to be open and should not
> be accessible from outside the cluster. It is important to protect this port
> because foreign commands can be sent directly to the IBM Worklight
> Analytics Platform through this port. In V6.2.0.1 and earlier, this port is
> open by default. In the latest interim fix, this port is closed by default. You
> can open this port by setting the `httpport.enabled` JNDI property to `true`.
> For more information, see "Properties and configurations" on page 1419.

Port 9600 - Transport Port
> This port is used for communication between nodes in a cluster. This port
> should be open to other nodes in the cluster for node communication to
> work properly. This port should also not be accessible from outside the
> cluster. This port is open by default.

These ports can be changed by using the following JNDI properties:
- `httpport`
- `transportport`

The following example shows how you can modify the ports.

```
<jndiEntry jndiName="analytics/httpport" value="9700" />
<jndiEntry jndiName="analytics/transportport" value="9800" />
```

Production deployment and clustering

Most of the clustering functionality and logic is handled by the IBM Worklight
Analytics Platform. It is not necessary to do any additional work to cluster the
application server that the IBM Worklight Analytics Platform is running on. The
application servers are only necessary to host the IBM Worklight Analytics
Platform and do not require special configuration for clustering.

Creating an IBM Worklight Analytics Platform cluster can be scoped down to the
following steps:

1. Configure the master node or nodes.
2. Set the number of shards.
3. Set the number of replicas.
4. Add a node to the cluster.
5. Point the new node to the master node or nodes.

It is important to fully understand how the clustering works for the IBM Worklight Analytics Platform before you create the cluster.

Clustering terminology:

Learn about clustering terminology for the IBM Worklight Analytics Platform.

Cluster
> A collection of one or more master and data nodes.

Master Node
> Coordinator of the cluster. Manages the distribution of shards and keeps track of all nodes in the cluster. There can be more than one master node. If a master node fails, then a new node that is marked as a master node is automatically elected as a new master node. The cluster cannot operate without at least one master node.

Data Node
> Workhorse of the cluster. Stores data and processes incoming search and index requests. A node can act as both a data node and a master node.

Shard Each data node stores data in a shard. For more information about shards, see "Understanding shards" on page 1404.

Replica shard
> Each shard can have any number of replicas. Replicas are used to ensure high availability in the case that a node is no longer available. For more information about replicas, see "Understanding replicas" on page 1408.

The following image shows a basic clustering topology:
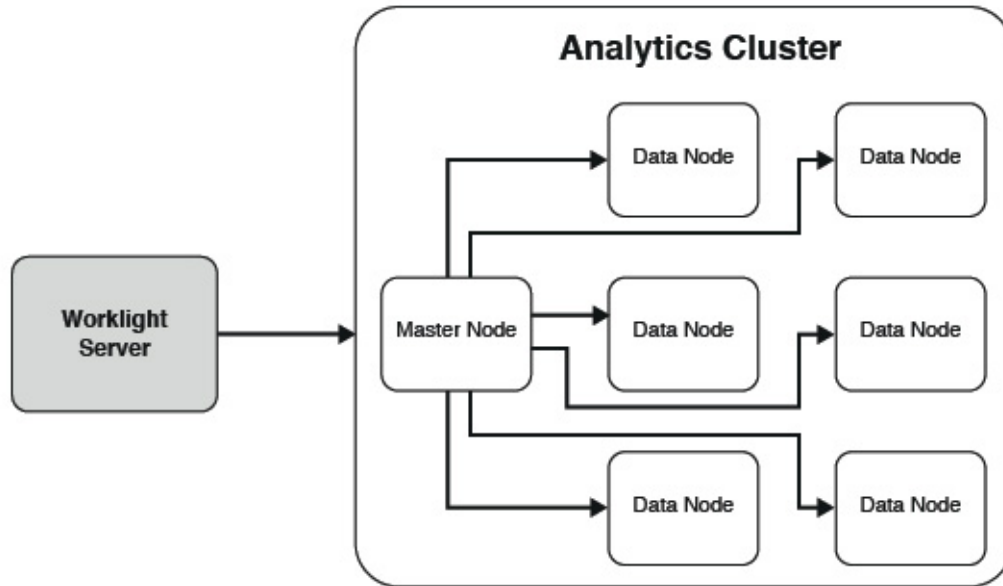
Note: Data can be forwarded from the Worklight Server to any node in the server. The Worklight Server does not have to point to a master node.

In development, a cluster that contains one node and one shard is sufficient. The single node acts as the master and data nodes.

In production, it is ideal to have multiple nodes that perform specific functions to ensure high availability and performance.

Understanding shards:

Learn about shards in the IBM Worklight Analytics Platform.

It is important to carefully consider setting the number of shards when you set up a cluster. The number of shards can be set only once, by the first node in the cluster. If the number of shards must be changed later, you must completely reindex all of the data that is stores in the IBM Worklight Analytics Platform.

Ideally, the number of shards is equal to the maximum number of nodes that the cluster eventually expands to. Because the maximum number of nodes that are needed is often unknown at installation time, it is a common practice to create more shards than needed.

The following images show how sharding works.

Here is a cluster with one node and six shards. Because there is only one node, all six shards live on the same node. The single node handles all requests and data processing.

Analytics Cluster

One Node   Six Shards
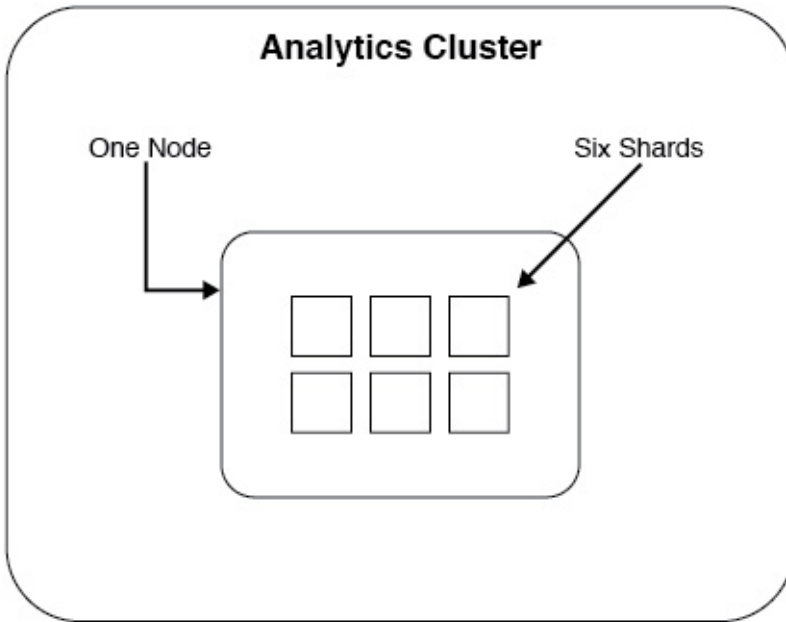
After several months of use, requests that are made to the cluster begin to perform poorly. It is determined that a single node is no longer adequate to handle processing for all of the incoming data. A new node is added to the cluster.

After a new node is added, the shards are automatically evenly distributed across all of the nodes.

Analytics Cluster

New Node
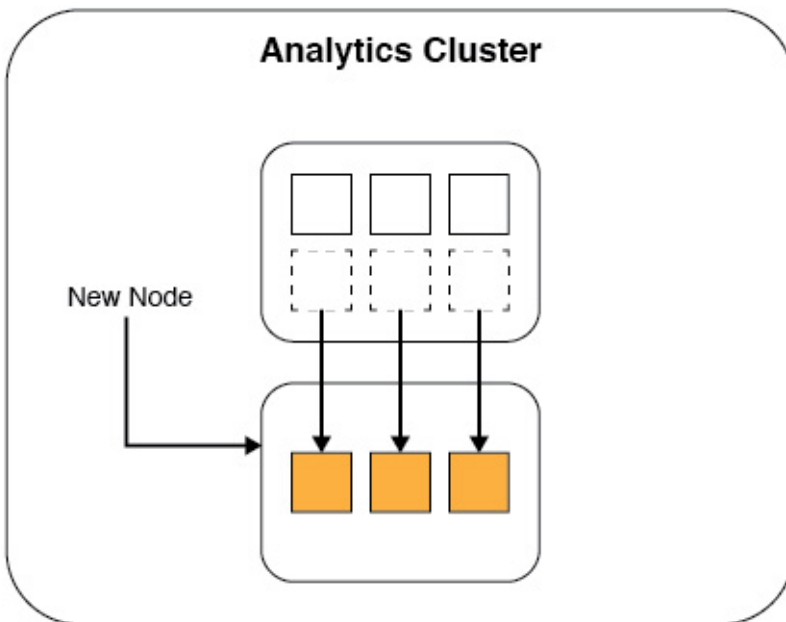
Now when a request comes in to either node, the request is forwarded to the node that has the shard that contains the data. The data indexing and processing is now split between the two nodes. Because the requests and data processing is now split between the nodes, the performance and response times improve.

Analytics Cluster

Requests split between the two nodes
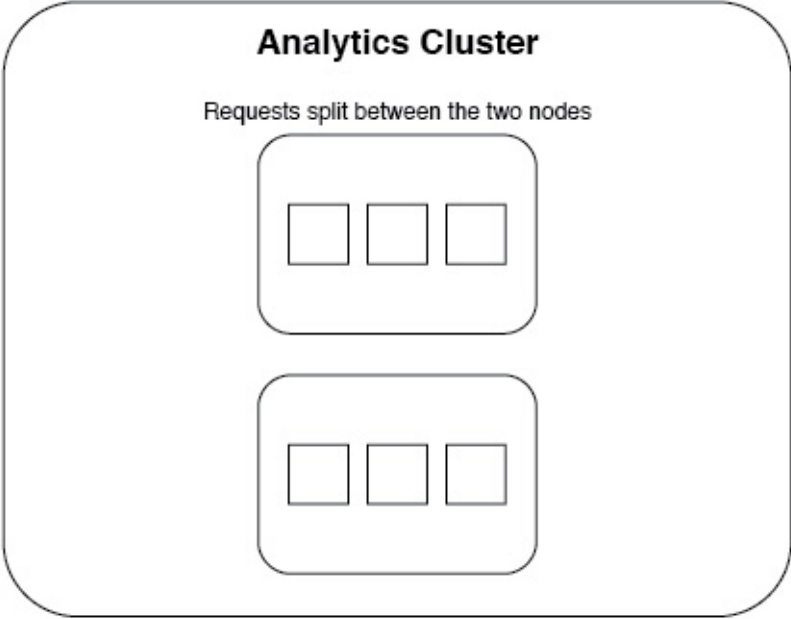
After several months, requests begin to slow down again, and it is determined that a third node is required.

Analytics Cluster
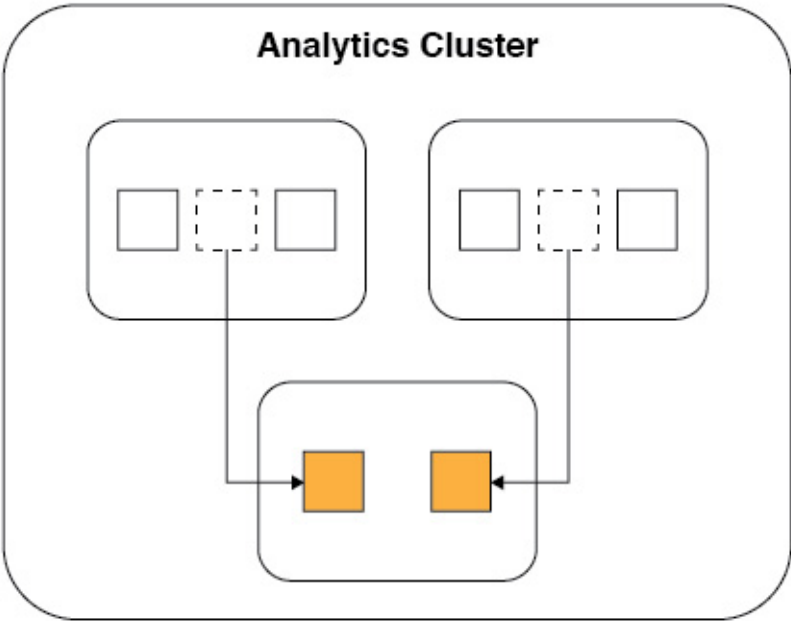
The shards are split between the three nodes.
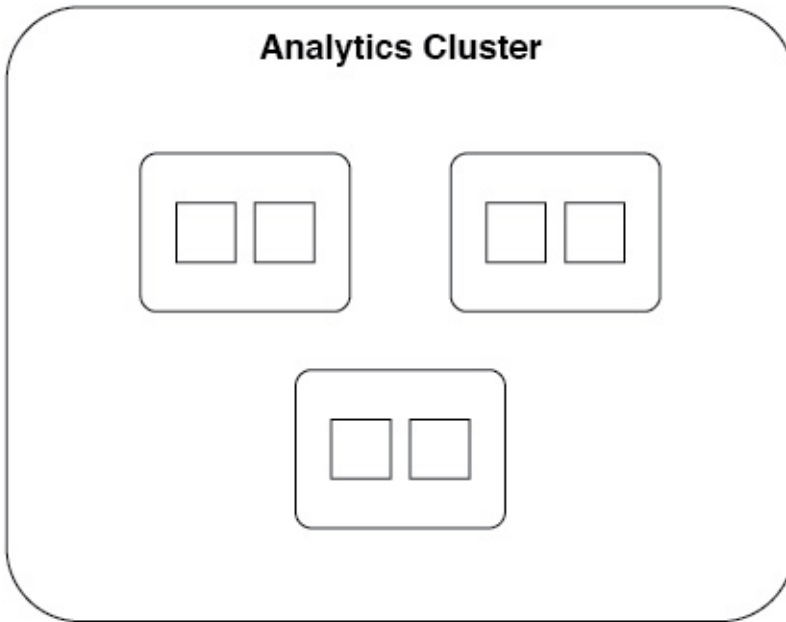
This process repeats itself until there are six nodes, which each contains one shard. It is now no longer possible to add more nodes because each shard contains only one node.
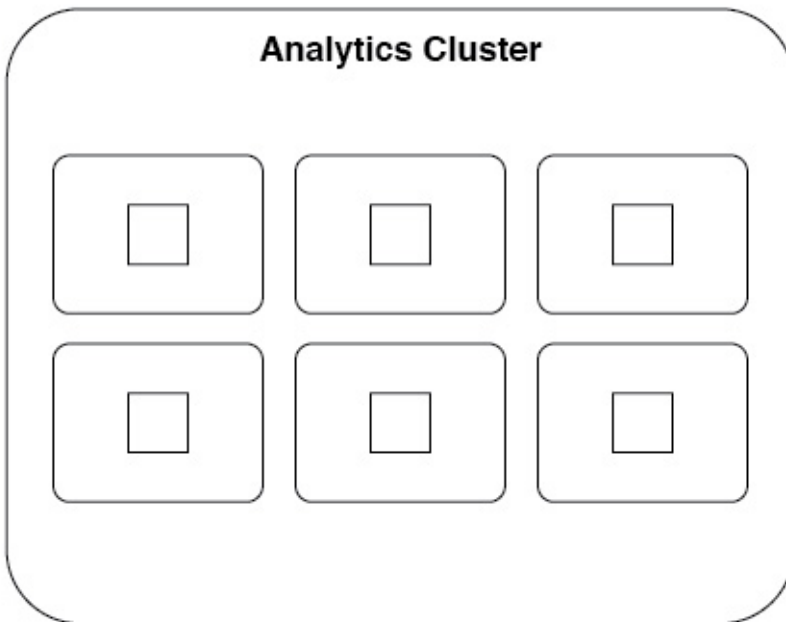
If it is determined that six nodes are no longer sufficient to handle the incoming data load, a new cluster must be set up. The data must then be reindexed with a larger shard limit.

It is important to understand that the distribution of the shards happens automatically. The only configuration that must be made for shards is specifying the number of shards at installation time.

A small performance hit comes with having more than one shard per node. Although this performance hit is often negligible, the cluster should not be configured with an arbitrarily large number of shards.

Understanding replicas:

Learn about replicas in the IBM Worklight Analytics Platform.

Shards contain the actual data that is sent from the Worklight Server. The master node keeps track of which shards are on which nodes so that it can evenly distribute incoming requests. Because of the way shards are distributed among nodes, performance can be increased by adding another node and allowing the shards to be distributed.

But what happens if a node fails? The data that was stored in the lost shards is no longer available. Incoming analytics data might no longer be indexable. Search requests for data on a particular shard fail. To increase shard availability to avoid these problems, you can create a replica of each shard. By using JNDI properties, you can tell the IBM Worklight Analytics Platform to create a specified number of replicas for each shard.

The following images show how replicas work.

Here is a cluster with one node, three shards, and one replica for each shard.
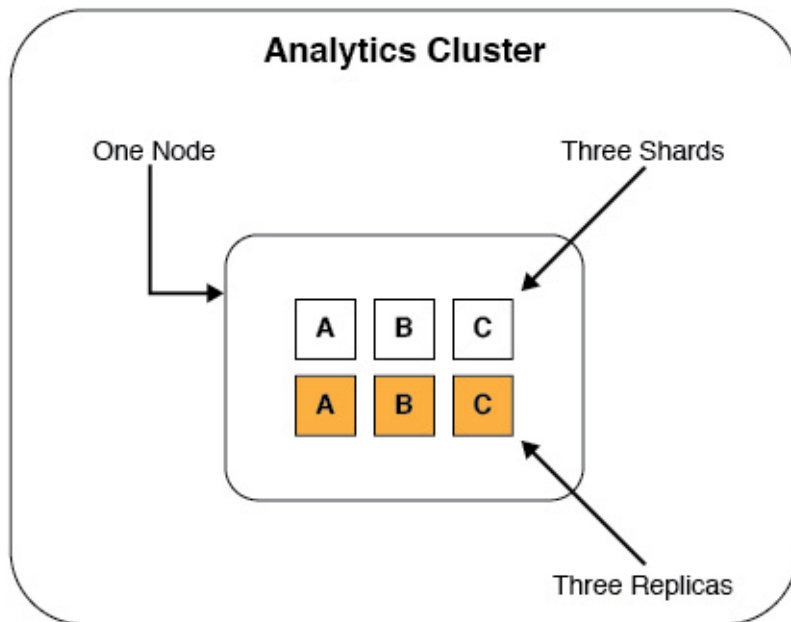
In this case, the replicas are redundant. Because there is only one node, having a replica exist on the same node does not accomplish anything. If the single node fails, the shards and replicas are all lost.

Now two more nodes are added to the cluster to improve performance:
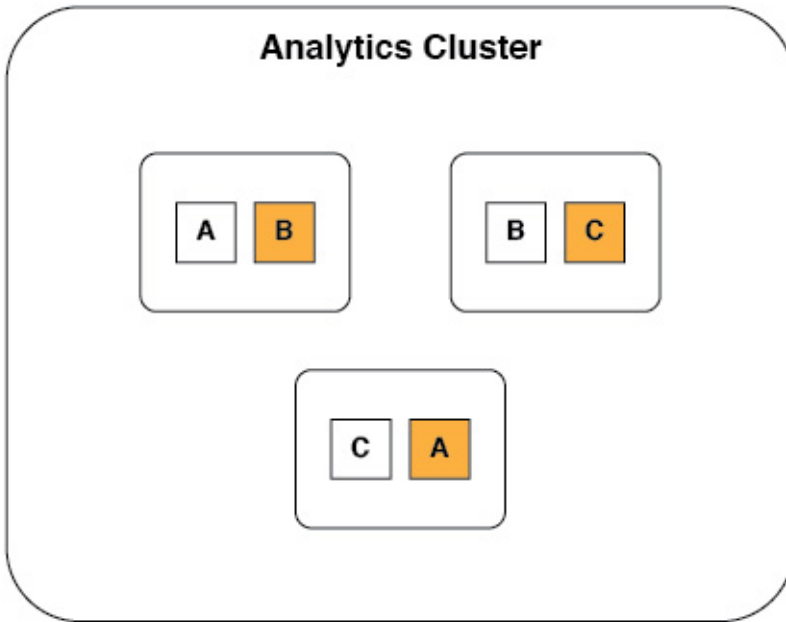
Notice that the shards and replicas are both automatically distributed evenly among the nodes. Now consider the scenario where a node fails due to a network or hardware issue.
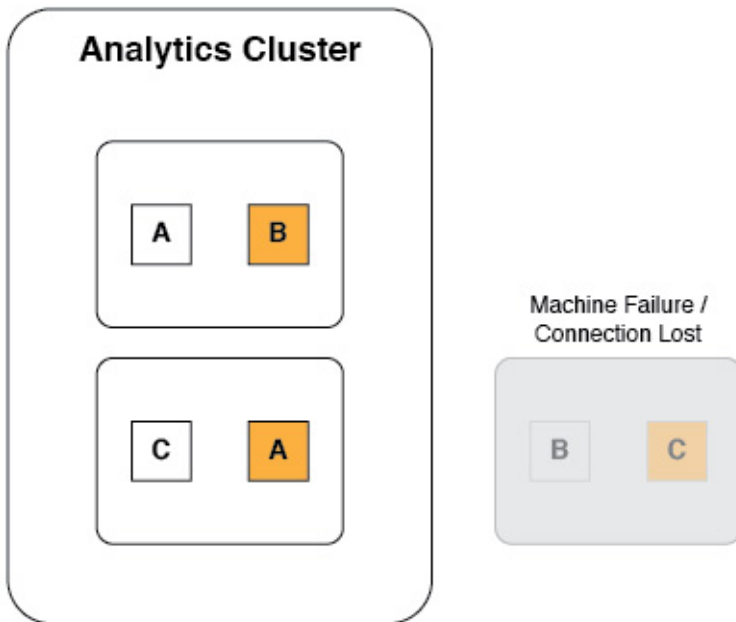
The analytics cluster can still operate normally because a replica of the lost shards still exists on one of the remaining nodes.
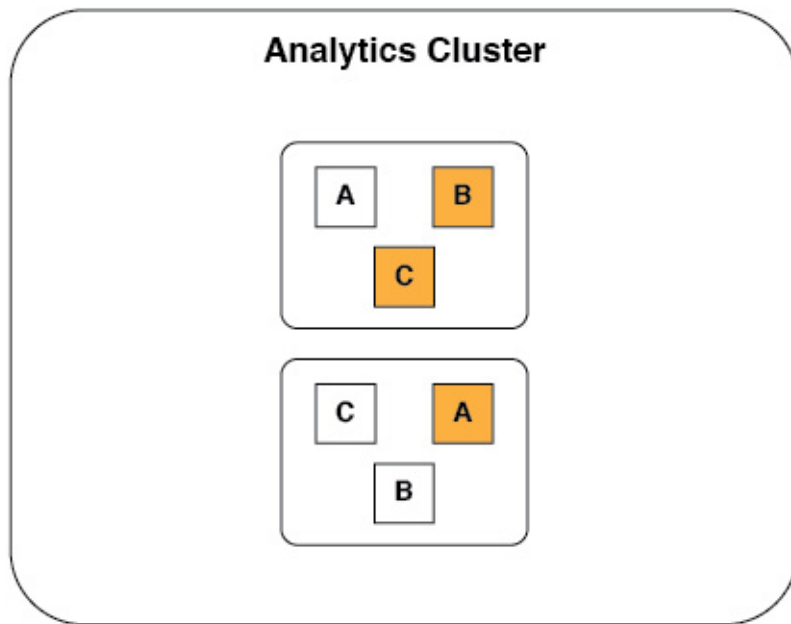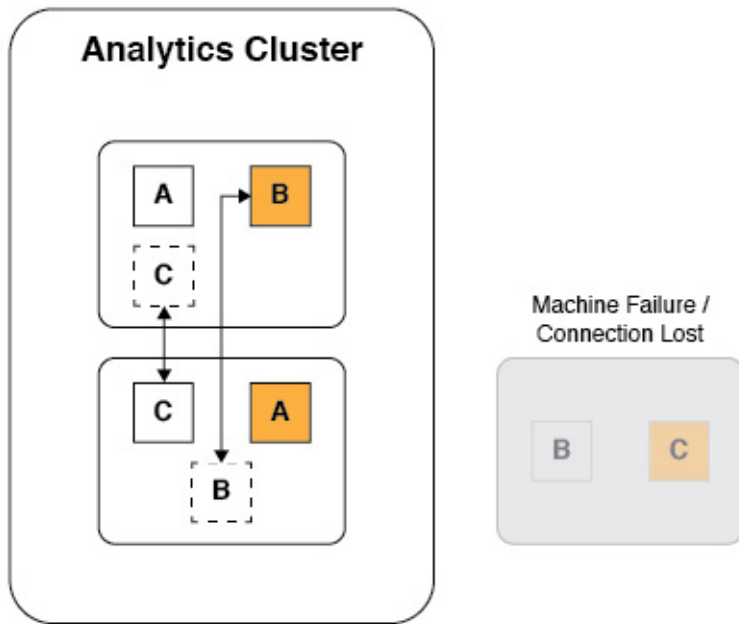
When the failed node comes back online and rejoins the cluster, the shards and replicas are again evenly distributed. The cluster returns to the state it was in before one of the nodes failed.

But what happens when two of the nodes fail simultaneously?
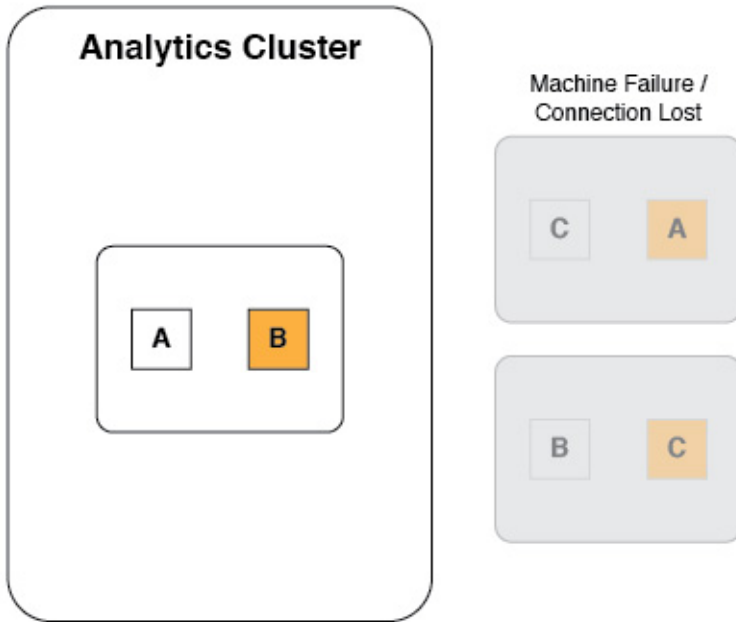
The cluster cannot operate normally. Even with one replica per shard, if two nodes were to fail, you would still lose information that was stored in the lost shards.

The answer to this problem is to use two replicas per shard.
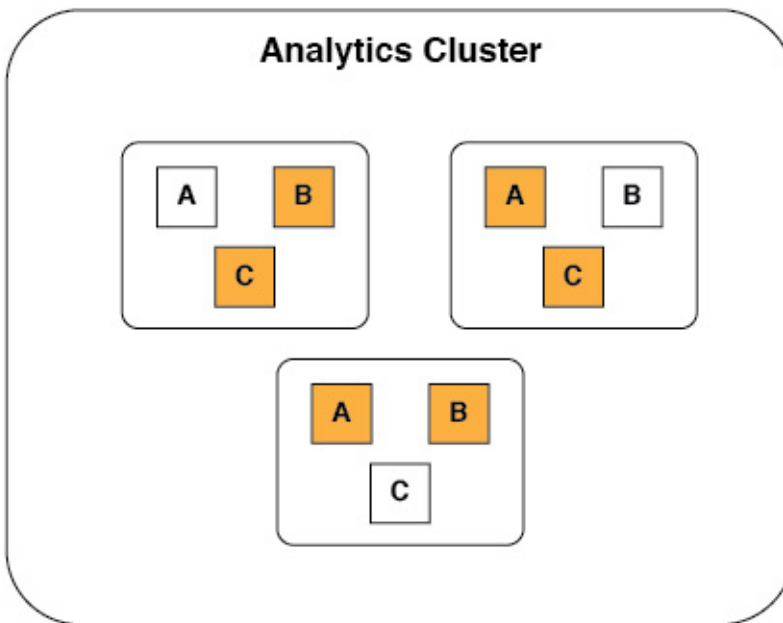
Now even when two nodes fail, all of the data is available and the cluster can still operate normally.

Having replica shards can also increase performance because an incoming request can be handled by either a primary or replica shard.

The ideal number of replicas for each shard varies based on several factors such as:

- Hardware limitations.

- Availability requirements.
- Clustering topology.

Setting up a production cluster:

You can set up a production cluster for operational analytics.

Before you begin

For a production cluster, do not run the IBM Worklight Analytics Platform on the same server as the Worklight Server. The IBM Worklight Analytics Platform uses a large amount of the computer's processor and memory resources. Each node should run on a separate server.

If you are deploying to a WebSphere Application Server cluster, also see "Deploying in a clustered WebSphere Application Server environment" on page 1414.

About this task

To set up your production cluster, follow these steps.

Procedure

1. Set the heap size for the application server. The heap size has a very significant impact on the performance of the IBM Worklight Analytics Platform. The heap size must be set on each application server that is hosting a node. The `-Xms` Java option sets the minimum heap size value. The `-Xmx` Java option sets the maximum heap size value. For example, to reserve 8 GB of memory for the IBM Worklight Analytics Platform, set the following Java options:

   ```
   -Xms8G -Xmx8G
   ```

   Note:
   - The minimum heap size for a production server is 8 GB.
   - Do not allocate more than half of the system memory to the application server that is running the IBM Worklight Analytics Platform.
   - Set the minimum and maximum heap size to the same value for an application server that is hosting the IBM Worklight Analytics Platform.

   For an IBM Worklight Analytics Platform that runs on a server with 32 GB of RAM, the optimal heap size is:

   ```
   -Xms16G -Xmx16G
   ```

2. Configure the first node in the cluster. The first node in the cluster is important because some properties can be set only by this node. It is important to ensure that the settings for the node are present before you start the application servers. The first node in a cluster must be a master node. A master node can also act as a data node.

   a. Configure the node to be a master node. Set the `nodetype` JNDI property to `master`.

      ```
      <jndiEntry jndiName="analytics/nodetype" value="master" />
      ```

      Note: By default, a node acts as a master and data node. If you want to create a node that acts as both a master and a data node, the node type does not need to be set.

b. Set the number of shards. Set the `shards` JNDI property to the number of shards you want.

```
<jndiEntry jndiName="analytics/shards" value="10" />
```

c. Set the number of replicas. Set the `replicas_per_shard` JNDI property to the number of replicas you want.

```
<jndiEntry jndiName="analytics/replicas_per_shard" value="2" />
```

After all of the JNDI properties are set, the application server can be started.

3. Add a node to an existing cluster.

a. Set the node type. Set the `nodetype` JNDI property to either `master` or `data`.

```
<jndiEntry jndiName="analytics/nodetype" value="master" />
<jndiEntry jndiName="analytics/nodetype" value="data" />
```

Note: By default, a node acts as a master and data node. If you want to create a node that acts as both a master and a data node, the node type does not need to be set.

Note: If you are deploying to a WebSphere Application Server cluster, do not set the `nodetype` JNDI property. All nodes in the WebSphere Application Server cluster must act as master and data nodes.

b. Provide a list of the master nodes. Set the `masternodes` JNDI property to a comma delimited list of host names for the master node.

```
<host>:<transport-port>,<host>:<transport-port>
```

The default transport port is 9600. You can change this value by using a JNDI property. For more information about the transport port, see "Ports that are used by the IBM Worklight Analytics Platform" on page 1402.

For example:

```
<jndiEntry jndiName="analytics/masternodes"
           value="master1.ibm.com:9600,master2.ibm.com:9600" />
```

4. Configure the `worklight.properties` file. Set the `wl.analytics.url` property to point to any of the nodes (can point to a master or data node).

```
wl.analytics.url=http://<hostname>:<port>/<context-root>/data
```

Results

You have set up a production cluster.

Note:

A node that is set as a pure data node cannot run on its own without a master node. The `masternodes` JNDI property must be set and must point to an active master node.

A node that is set as a pure master node cannot store data. Any attempts to store or view data from a master node fails until a data node connects to it.

Data nodes are active, which makes them more prone to network and hardware failures. If a data node fails, the cluster can still operate normally if the other data nodes are alive. If the master node or nodes fail, then the cluster cannot operate. Data nodes can communicate with only each other through master nodes. This behavior is why it is important to consider having separate master nodes that can run on servers with fewer chances for failure.

The same version of Java must be installed on all nodes in the cluster. Using different versions of Java for different nodes causes the cluster to fail.

Deploying in a clustered WebSphere Application Server environment:

You can deploy the Analytics Platform in a clustered WebSphere Application Server environment.

Before you begin

Note: In previous topics, the term *node* is used as a general term to define a separate machine that runs an instance of the Analytics Platform. The term *node* is used here to identify a node in a WebSphere Application Server cluster.

About this task

All clustering is handled by the Analytics Platform regardless of the topology that is configured in WebSphere Application Server. You must determine which machines in the cluster you want to be the *master nodes* and define them through a JNDI property. The list of master nodes are passed down to the WAR file, where the distribution of analytics data within the cluster is handled.

To use the Analytics Platform in a clustered environment on WebSphere Application Server, follow these steps.

Procedure

1. Identify the machines in the cluster that you want to be the master nodes and record their IP addresses or host names.
2. Deploy the analytics WAR file to the WebSphere Application Server cluster. Do not start the web application yet.
3. Set the JNDI property for `masternodes` to a comma-separated list of the nodes in the cluster that you want to be the host name. The following example shows possible values for the JNDI property:

   `<hostname>:<port>,<hostname>:<port>,<hostname>:<port>`

   `192.168.1.32:9600,192.168.1.8:9600`

   `clusterhost1:9600,clusterhost2:9600,clusterhost3:9600`

   Note: The *port* value is the transport port, which by default is 9600. You can change this port through the JNDI property. For more information, see "Ports that are used by the IBM Worklight Analytics Platform" on page 1402. The following image shows the 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
analytics	analytics.war,WEB-INF/web.xml	analytics/nodetype	String	[OPTIONAL] Defines the node type. Valid values are "master" and "data". If this JNDI property is not set, then by default, the node will act as a master node and a data node.	
analytics	analytics.war,WEB-INF/web.xml	analytics/shards	String	[OPTIONAL] The number of shards that the cluster will create. This value can only be set by the first node in a cluster. Default: 5	
analytics	analytics.war,WEB-INF/web.xml	analytics/replicas_per_shard	String	[OPTIONAL] The number of replicas for each shard in the cluster. This value can only be set on the first node in a cluster. Default: 1	
analytics	analytics.war,WEB-INF/web.xml	analytics/masternodes	String	[OPTIONAL] A comma delimited string containing the hostname and ports of the master nodes (hostname:transport-port,hostname:transport-port)	9.122.123.134:9600

4. Set the remaining JNDI properties. For more information about JNDI properties, see "Properties and configurations" on page 1419.

5. Open the analytics console page on each cluster. When the console is accessed, each node in the cluster establishes a connection with each node listed in the `masternodes` list.

Results

You deployed the Analytics Platform in a clustered WebSphere Application Server environment.

Note: When you install analytics on WebSphere Application Server, a profile can contain multiple servers. Each server can have multiple analytics WAR files. By default, each of these WAR files points to the same directory to store analytics data. Normally, a JNDI property is used to change the location of the directory that is used to store analytics data. However, in this case, each server shares the same JNDI property. Because of this scenario, you must use WebSphere Application Server variables to define the location of the data directory when you set the JNDI property for the analytics data folder. The following example shows how you can set the JNDI property with the variables:

`${USER_INSTALL_ROOT}/MFPAnalyticsDir/${WAS_SERVER_NAME}/AnalyticsData`

Performance tuning:

Learn about performance tuning for the IBM Worklight Analytics Platform.

Worklight throughput-tuning

You can tune the amount of data that is held on the Worklight Server to balance the risk of data loss against the risk of overloading the IBM Worklight Analytics Platform. When Analytics data is sent from the client, it is bundled together and placed in a queue on the Worklight Server. When a queue is full on the Worklight Server, it posts that data to the Analytics Platform. Then, the queue empties.

You can use the follow two parameters in the JNDI configuration for throughput-tuning:
- `wl.analytics.queues`
- `wl.analytics.queue.size`

The `wl.analytics.queues` parameter determines the maximum number of queues that the Worklight Server holds in memory. If all of the queues fill up before they post to the Analytics Platform, the Worklight Server drops data that is received from the client until a queue empties.

The `wl.analytics.queue.size` parameter is the number of individual elements that each queue can hold. Adjustment of these parameters affects:
- Memory that is used by the server.
- Frequency of POSTs to the IBM Worklight Analytics Platform.

The following image shows the data accumulation on the Worklight Server:
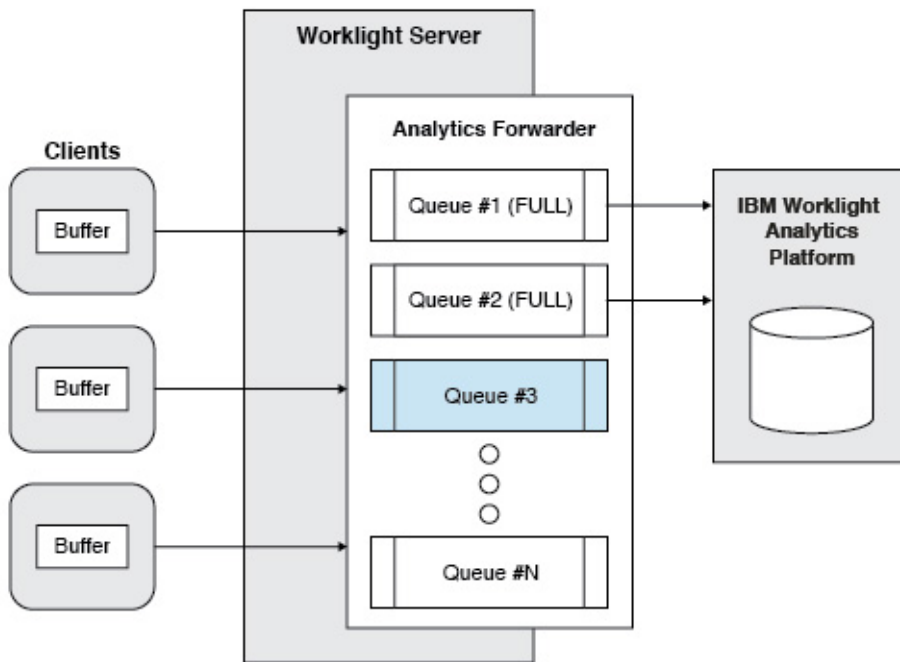
The number of individual analytics events that the server holds at one time is `wl.analytics.queues * wl.analytics.queue.size`. Take this fact into consideration when you define these two parameters. If you set them too low, large amounts of analytics data can be dropped if the server is unusually busy. If you set them too high, too much memory can be used on the Worklight Server.

Java virtual machine (JVM) swapping

The underlying technology that is used by the IBM Worklight Analytics Platform is called Elasticsearch. Elasticsearch performs poorly when the JVM starts swapping. To ensure that the JVM never swaps, the following JNDI property can be set to `true`:

```
<jndiEntry jndiName="analytics/bootstrap.mlockall" value="true" />
```

Administration
Learn about the administrative aspects of the IBM Worklight Analytics Platform.

Multi-tenancy:

Learn about multi-tenancy in the IBM Worklight Analytics Platform.

Several Worklight Servers can be configured to send analytics data to the same analytics cluster. All of the data is indexed together, which means that all charts and queries that are performed on the analytics server reflect data that was sent from every Worklight Server. If you want to use the same analytics cluster for multiple Worklight Servers, but also want the data to be indexed separately so that it can be searched and viewed separately, you can add a tenant.

Servers can be configured to send their data to a new tenant on the analytics cluster so that the data can be viewed separately, even though all of the data lives on the same cluster.

To forward data to a different tenant, append the following format to the `wl.analytics.url` property on the Worklight Server:

`?tenant=<tenant-name>`

For example, if you want to send data to the default tenant, set the `wl.analytics.url` property as follows:

`wl.analytics.url=http://host.ibm.com/analytics/data`

If you want to send data to a new tenant that is named `test`, set the `wl.analytics.url` property as follows:

`wl.analytics.url=http://host.ibm.com/analytics/data?tenant=test`

To view the analytics data for a specific tenant, append the same format to the URL for the analytics console:

`http://host.ibm.com/analytics/console?tenant=test`

Data purging:

Learn about data purging in the IBM Worklight Analytics Platform.

By default, data that is stored in the Analytics Platform is not automatically deleted. To enable automatic purging of data, the time to live (TTL) property must be set for each data type.

The TTL for analytics data types that are stored in the Analytics Platform can be set by using JNDI properties. For more information about analytics data types, see "Operational analytics" on page 1397.

The following table shows the TTL properties:

Table 319. TTL properties for purging data that is stored in the Analytics Platform. This table lists TTL property names and description for purging data that is stored in the Analytics Platform.

Property Name	Description
`app_activities_ttl`	Time to live for app activities, such as session starts, adapter hits, and network hits.
`notification_activities_ttl`	Time to live for notification activities, such as push notifications.
`client_logs_ttl`	Time to live for client logs, such as client-side captured logs, and stack traces.
`server_logs_ttl`	Time to live for server logs.

Note: All JNDI properties must be preceded with the `analytics/` string. For more information about JNDI properties, see "JNDI properties" on page 1420.

By default, the format for the TTL is in milliseconds. TTL can also be set by using a number followed by a character that represents the time interval:
- d (days)
- m (minutes)
- h (hours)
- ms (milliseconds)
- w (weeks)

The following example shows how to set the app activities data TTL to one day in milliseconds:

```
<jndiEntry jndiName="analytics/app_activities_ttl" value="86400000" />
```

The following example shows how to set the client logs data TTL to five days:

```
<jndiEntry jndiName="analytics/client_logs_ttl" value="5d" />
```

The following example shows how to set the server logs TTL to one week:

```
<jndiEntry jndiName="analytics/server_logs_ttl" value="1w" />
```

Note: The TTL properties are not applied to data that already exists in the Analytics Platform. You must set the TTL properties before you add data.

Exporting raw reports:

Learn about exporting raw reports in the IBM Worklight Analytics Platform.

Raw analytics data can be exported from the IBM Worklight Analytics Platform for the following types of data:
- Application session data
- Adapter invocation data
- JSONStore operation data

This data can be exported through a REST API that is exposed by the IBM Worklight Analytics Platform. Currently, the supported export formats include JSON and CSV.

Exporting application sessions

The following example shows the format for exporting analytics data for application sessions:

```
/export/{tenant}/sessions/{days}/{gadgetName}/{gadgetVersion}/{env}/{model}/{os}/{limit}/{offset}
```

If the analytics console is hosted here:

```
http://hostname.ibm.com:9080/analytics
```

Then, session data can be exported by invoking by following link, and passing an Accept header that specifies the desired content mime type:

```
http://hostname.ibm.com:9080/analytics/data/export/{tenant}/sessions/{days}/{gadgetName}/{gadgetVers
```

For example, by using the curl command-line tool:

```
curl -H "Accept: application/csv"
"http://hostname.ibm.com:9080/analytics/data/export/worklight/sessions/30/TestApp/*/iphone/*/7.0/100/0"
```

This previous curl command exports data from the tenant that is named `worklight` for all versions of the application that is called `TestApp` for the last 30 days. It returns only data for the iPhone environment, for all models of the iPhone, and only for iOS 7.0. It returns the first 100 results that are found and start with the first result (limit = 100, offset = 0).

Note:
- The default tenant is `worklight`. If you did not configure a specific tenant for the IBM Worklight Analytics Platform, then you should use `worklight`.

Exporting adapter invocations

The following example shows the format for exporting analytics data for adapter invocations:

```
/export/{tenant}/adapters/{days}/{adapter}/{procedure}/{gadgetName}/{gadgetVersion}/{env}/{model}/
```

For example, by using the curl command-line tool:

```
curl -H "Accept: application/csv"
"http://hostname.ibm.com:9080/analytics/data/export/worklight/adapters/10/UploadAdapter/uploadProcedure/TestApp/1.0/android/nexus/4.4/10/0"
```

Exporting JSONStore operation data

The following example shows the format for exporting analytics data for JSONStore operation data:

```
/export/{tenant}/jsonstore/{days}/{gadgetName}/{gadgetVersion}/{env}/{model}/{os}/{collection}/{op
```

For example, by using the curl command-line tool:

```
curl -H "Accept: application/csv"
"http://hostname.ibm.com:9080/analytics/data/export/worklight/jsonstore/100/*/*/iphone/*/*/people/add/100/50"
```

Properties and configurations

Learn about the properties and configurations that are used for configuring the Worklight Server and IBM Worklight Analytics Platform.

Worklight properties

These properties can be set on the Worklight Server in the `worklight.properties` file. The server must be restarted for these properties to take effect.

Note: All properties in the `worklight.properties` file can also be set by using JNDI properties. For more information about JNDI properties, see "Configuring a Worklight project in production by using JNDI environment entries" on page 1182.

The following table shows the IBM Worklight Analytics Platform properties:

Table 320. IBM Worklight Analytics Platform properties. This table lists the property names, default values, and descriptions for the IBM Worklight Analytics Platform.

Property Name	Default Value	Description
`wl.analytics.url`	None.	Required. The URL that is exposed by the IBM Worklight Analytics Platform that receives incoming analytics data. Example: `http://<hostname>:<port>/<context-root>/d`

Table 320. IBM Worklight Analytics Platform properties (continued). This table lists the property names, default values, and descriptions for the IBM Worklight Analytics Platform.

Property Name	Default Value	Description
wl.analytics.logs.forward	true	When the property is set to true, server logs that are recorded on the Worklight Server are captured and forwarded to the IBM Worklight Analytics Platform.
wl.analytics.username	None.	The user name that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
wl.analytics.password	None.	The password that is used if the data entry point for the IBM Worklight Analytics Platform is protected with basic authentication.
wl.analytics.queues	20	Maximum number of concurrent queues that are used to buffer data before it is forwarded to the IBM Worklight Analytics Platform. For more information, see "Worklight throughput-tuning" on page 1415.
wl.analytics.queue.size	10	Size of each queue that is used to buffer data before it is forwarded to the IBM Worklight Analytics Platform. For more information, see "Worklight throughput-tuning" on page 1415.

JNDI properties

JNDI environment properties can bet set on the application server. For a clustered environment, some properties can be set only on the first node in the cluster. For more information, see "Setting up a production cluster" on page 1412. The Analytics runtime web application must be restarted for any changes in these properties to take effect. It is not necessary to restart the entire application server.

All JNDI properties are namespaced with analytics/.

The following example shows how to set the datapath JNDI property in Liberty:

```
<jndiEntry jndiName="analytics/datapath" value="/opt/IBM/analytics/data" />
```

The following example shows how to set the datapath JNDI property in Tomcat:

```
<Environment name="analytics/datapath"
          value="/opt/IBM/analytics/data"
          type="java.lang.String"
          override="false" />
```

Note: For Tomcat, the JNDI property must include the `override="false"` attribute to work properly.

The following table shows the JNDI properties:

Table 321. JNDI properties for the IBM Worklight Analytics Platform. This table lists the JNDI property names, default values, and descriptions for the IBM Worklight Analytics Platform.

Property Name	Default Value	Description
nodetype	None.	Defines the node type. Valid values are `master` and `data`. If this JNDI property is not set, then the node acts as a master node and a data node by default.
shards	5	The number of shards that the cluster creates. This value can be set only by the first node in a cluster. This value can never be changed after the first node in the cluster starts.
replicas_per_shard	1	The number of replicas for each shard in the cluster. This value can be set only by the first node in a cluster.
masternodes	None.	A comma-delimited string that contains the hostname and ports of the master nodes. For more information about this property, see "Setting up a production cluster" on page 1412.
clustername	worklight	Name of the cluster. Set this value if you plan to have multiple clusters and want to uniquely identify them.
nodename	Randomly generated.	Name of a node in a cluster. A node that joins a cluster randomly generates a name to uniquely identify itself. You can specify your own name by using this property.
datapath	./analyticsData	The path that analytics data is saved to on the file system. By default, a folder that is named `analyticsData` is created.
settingspath	None.	Specifies the path to an extra settings file. For more information, see "Elasticsearch properties" on page 1422.

Table 321. JNDI properties for the IBM Worklight Analytics Platform (continued). This table lists the JNDI property names, default values, and descriptions for the IBM Worklight Analytics Platform.

Property Name	Default Value	Description
`transportport`	9600	Port that is used for node-to-node communication. For more information, see "Ports that are used by the IBM Worklight Analytics Platform" on page 1402.
`httpport`	9500	Port that is used for HTTP communication to the IBM Worklight Analytics Platform. For more information, see "Ports that are used by the IBM Worklight Analytics Platform" on page 1402.
`http.enabled`	`true` (V6.2.0.1 and earlier) `false` (latest interim fix)	Controls whether Elasticsearch can be queried directly by using the HTTP Port. If `false`, the port that is specified by the JNDI value `httpport` is not opened, and Elasticsearch is not accessible by using HTTP.
`app_activities_ttl`	None.	The TTL for automatic deletion of app activities data.
`notification_activities_ttl`	None.	The TTL for automatic deletion of notification activities data.
`client_logs_ttl`	None.	The TTL for automatic deletion of client logs data.
`server_logs_ttl`	None.	The TTL for automatic deletion of server logs data.

Elasticsearch properties

Elasticsearch is the underlying technology that is used by the IBM Worklight Analytics Platform. Elasticsearch provides several extra properties for performance tuning. The JNDI properties that are exposed and documented here are abstractions around the properties that are provided by Elasticsearch. Normally, these properties are set in a custom settings file.

If you are familiar with Elasticsearch and the format of its properties files, you can specify the path to the settings file by using the `settingspath` JNDI property:

```
<jndiEntry jndiName="analytics/settingspath"
           value="/home/system/elasticsearch.yml" />
```

All properties that are provided by Elasticsearch can also be set by using a JNDI property with the `analytics/` string added before the property name. For example,

`threadpool.search.queue_size` is a property that is provided by Elasticsearch that is used for performance tuning. This property can be set by using a JNDI property as follows:

```
<jndiEntry jndiName="analytics/threadpool.search.queue_size" value="100" />
```

Reports database

IBM Worklight Foundation provides an extensible mechanism for enterprises to use to integrate reporting tools with IBM Worklight Foundation.

IBM Worklight Foundation provides raw data reports and a number of device reports that are aggregated from the raw data report table. IBM Worklight Foundation 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 1395.

Note: Enabling the BIRT feature is redundant if you already use the IBM Worklight Analytics Platform.

IBM Worklight Foundation provides three reporting mechanisms:

Raw data feeds
IBM Worklight Foundation 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 1424.

Device usage reports
IBM Worklight Foundation 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 Foundation with an OLAP system. For more information, see "Device usage reports" on page 1428.

Note: Device usage reports are functional only in IBM Worklight Foundation Customer Edition and IBM Worklight Foundation Enterprise Edition.

BIRT reports
IBM Worklight Foundation 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 1430.
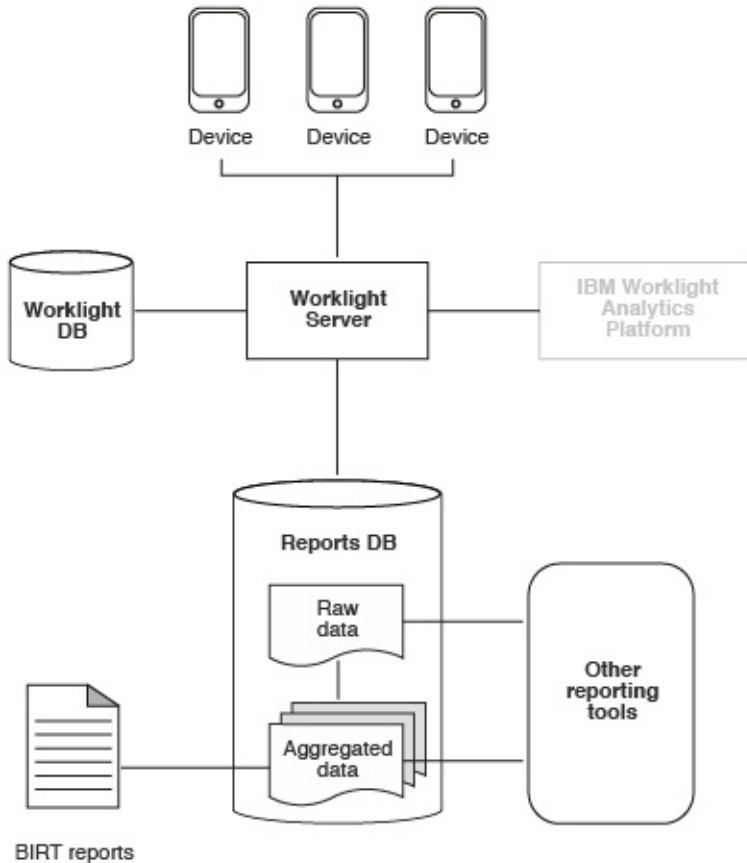
Figure 213. High-level overview of the reports architecture

The reports architecture diagram shows how the raw data feed comes from three devices into the Worklight Server and then into the IBM Worklight Foundation 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 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

"Worklight database setup for development mode" on page 1173. For information about creating databases and schemas in a production environment, see "Creating and configuring the databases" on page 1110.

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 a Worklight project in production by using JNDI environment entries" on page 1182.

5. Restart your application server.

   The `app_activity_report` table of the raw data database is populated with data as you use your applications and adapters.
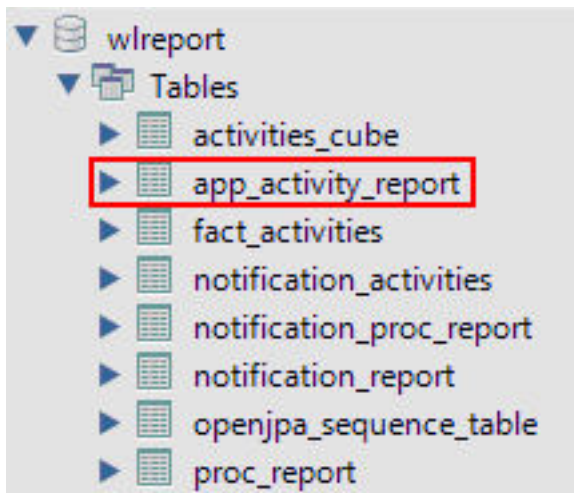
The raw data `app_activity_report` table contains the following information:

Column	Description
ACTIVITY_TIMESTAMP	UTC time of entry
GADGET_NAME	Worklight Application name

Column	Description
`GADGET_VERSION`	Application version
`ACTIVITY`	Activity type
`ENVIRONMENT`	Application environment name (iPhone, Android, and so on)
`SOURCE`	User identifier
`ADAPTER`	Worklight adapter name
`PROC`	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.
`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

Activity	Description
Logout	User logout
Event	An event handler was called

In addition to predefined activity types, custom activities can be logged by using **WL.Client.logActivity("custom-string")** APIs.

When the activity is Event, the reporting information comes from the event device context instead of WL.Server.getClientDeviceContext. Also, when the activity is Event the **PROC** column gives the name of the event handler function that was called.

Important: Worklight raw data feed can increase rapidly. The data is typically used by a BI system such as Cognos® or Business Objects. It is the administrator's responsibility to purge built-in tables periodically. For example, the following commands delete Oracle database rows that are more than 30 days old from the activities_cube and app_activity_report tables. For other databases such as MySQL, 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 of Worklight Server project databases" on page 184.

In addition to the app_activity_report table, the raw data engine also populates the notification_report table. This raw data table contains information about notifications that are sent from SMS event sources.
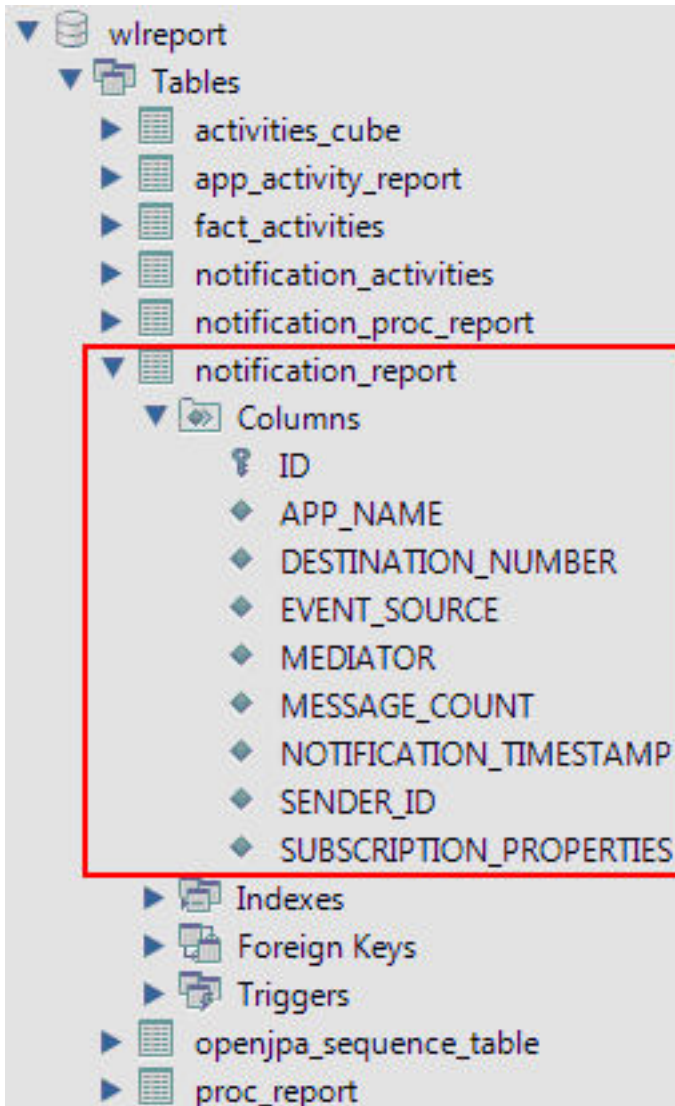
Device usage reports

For simpler and faster access to the reports data, Worklight Server runs an analytics data processor task at a default time interval of every 24 hours.

The analytics data processor task retrieves raw entries for the specified time interval from the `app_activity_report` table and processes them to populate the `fact_activities` table.

Note: The `fact_activities` table is only populated with usage data from hybrid and native applications from actual devices. Usage data from Worklight mobile web applications that are running on actual devices or from a browser, such as when you are using preview, is not populated into this table.

The fact_activities table contains a total activity count (number of logged actions) per application, application version, device, and environment. The fact_activities data is also processed and put into the activities_cube table. This table has the same structure as the fact_activities table and only contains records for the last 30 days.
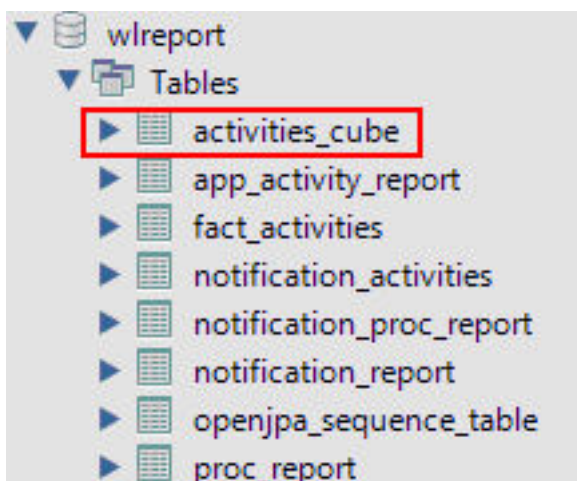
Each time the data processing is done, a time stamp is added to a `proc_report` table with the processing result (time stamp and number of processed entries).
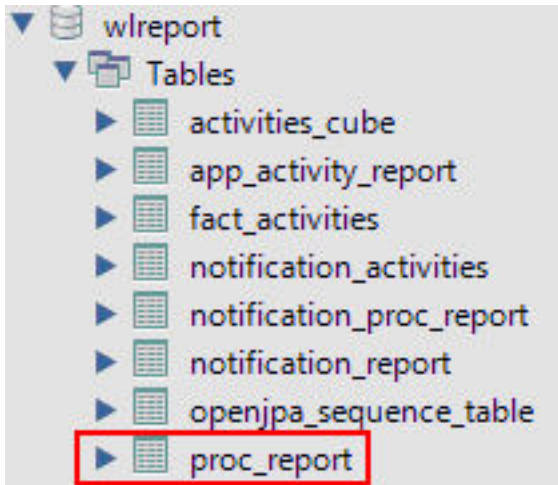
In addition, `notification_report` table data is also processed to populate the `notification_activities` table with consolidated data. The table is populated in the same way as the `fact_activities` table. Every time the `notification_report` table data is processed, an entry is added to the `notification_proc_report` table, which is similar to the `proc_report` table.
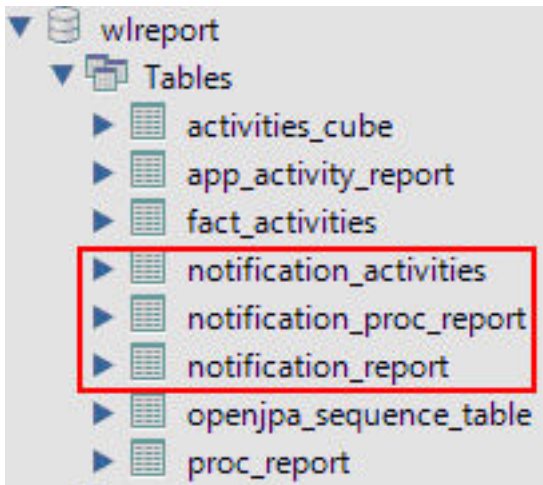
The processing interval can be modified by adding the following property to your `worklight.properties` file and setting the required interval in seconds.

```
# Default interval value for analytics processing task
wl.db.factProcessingInterval=86400
```

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 Foundation generates raw reports, which are stored in an app_activity_report table. IBM Worklight Foundation also includes device usage reports, which are aggregations of data from the app_activity_report, and are described in "Device usage reports" on page 1428 and "Using raw data reports" on page 1424. 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 Foundation 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 Foundation 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 Consumer Edition and IBM Worklight Enterprise Edition include the following predefined BIRT reports:

Table 322. 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 322. 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	c7c592dc-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 214. 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 1439
- 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 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 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`.

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.
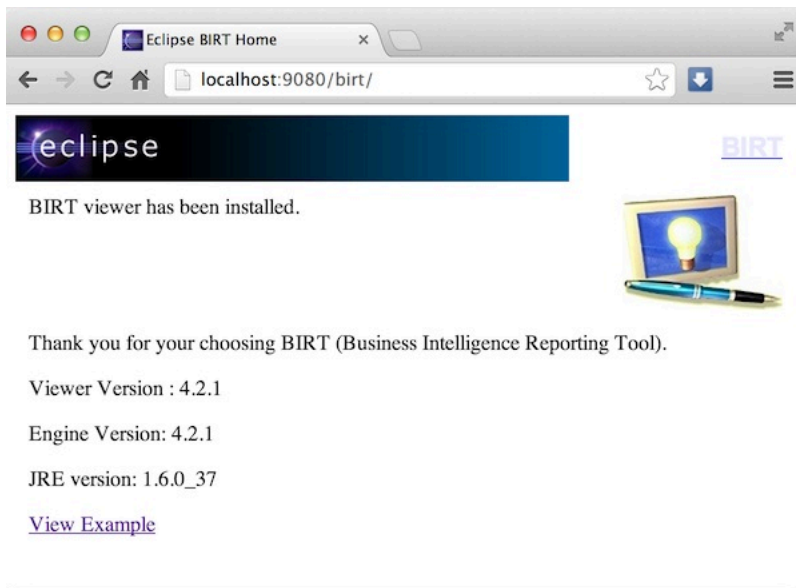
15. Click **View Example** link.
16. If you see the following error message, refresh your page.
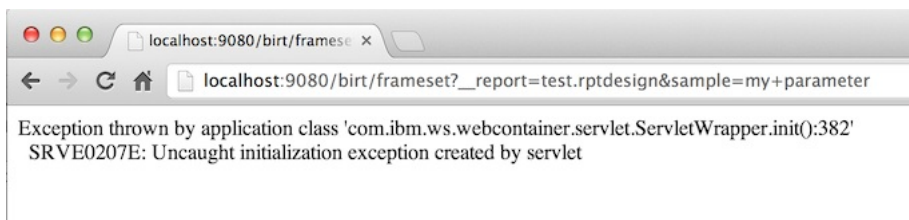
17. The BIRT Viewer sample report appears.
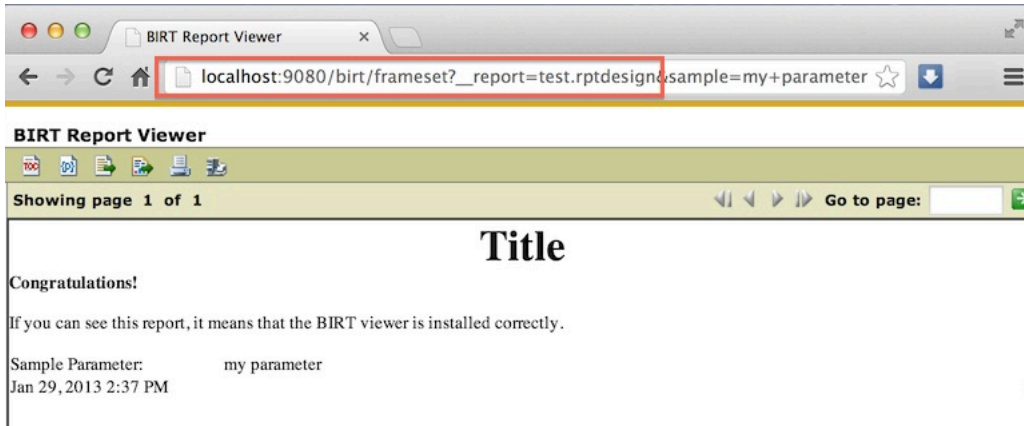
Note `test.rptdesign` in the page URL. You can replace this text with the name of other **rptdesign** files, as shown here for example:
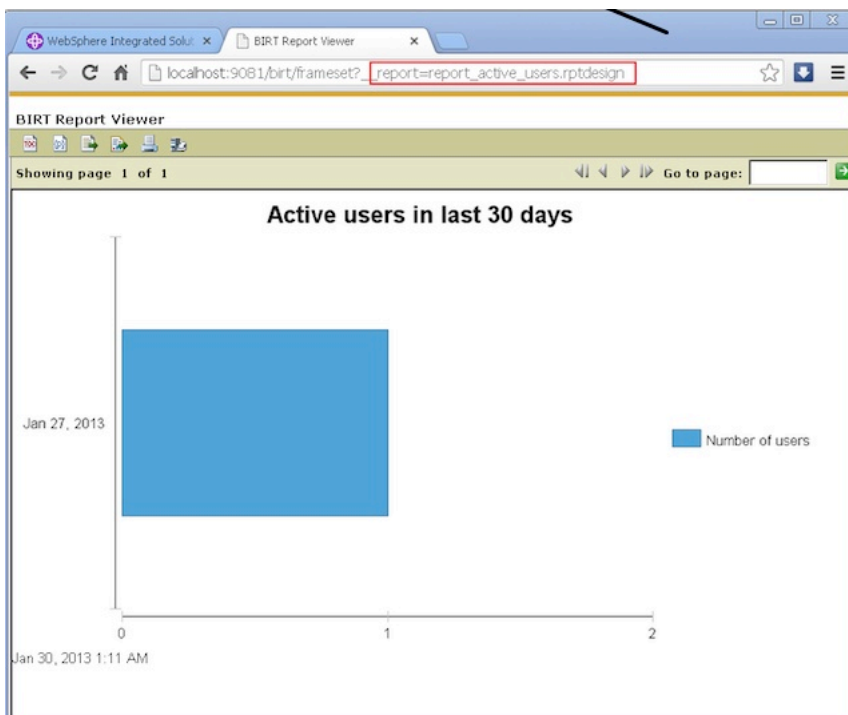
Installing BIRT on WebSphere Application Server full profile

Complete these steps to install Business Intelligence Reporting Tools (BIRT) on WebSphere Application Server full profile.

Procedure

1. Download the BIRT package and extract the contents.
2. From the folder `birt-runtime-version\WebViewerExample\WEB-INF\lib`, delete (or remove) the following packages:
   - `org.apache.xerces.jar`
   - `org.apache.resolver.jar`
   - `org.apache.serializer.jar`
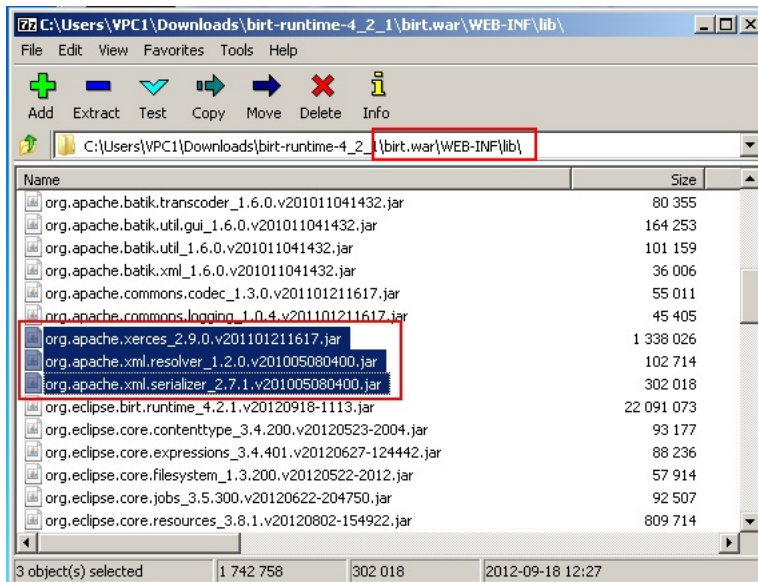
Figure 215. Deleting three files

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.

Chapter 14. Monitoring and mobile operations **1437**

Procedure

1. Ensure that the <configureapplicationserver> invocation has the inner element <reports todir=*"web applications directory"/>*. See "Ant tasks for installation of Worklight runtime environments" on page 1146 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 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 Foundation 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 Foundation can provide you with a more integrated interface, for both IBM Worklight Foundation 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 Foundation 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 Foundation, 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**.
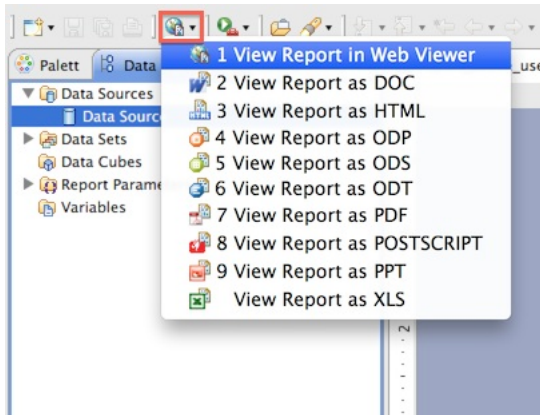
2. Select the output format for your report
3. View the report.
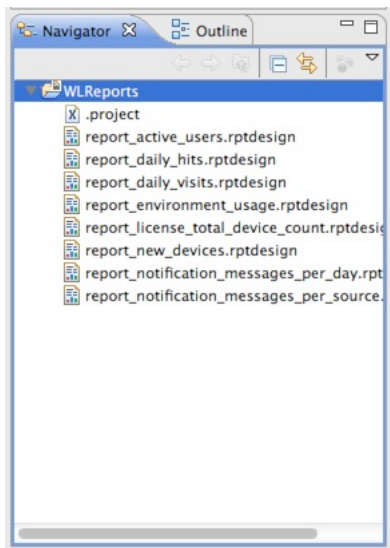
Notification reports database schema
IBM Worklight Foundation uses a database schema to store the notification reports data derived from the raw data.

Figure 216. 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.

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 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 IBM 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 that is called `Devices`. With this tab, the 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.

Figure 217. User or friendly name search

The `Status` column contains the current access rights of the device. Any device with the column marked as "Stolen", "Lost", or "Disabled" is not allowed to access Worklight Server. The "Expired" status is used only for licensing purposes. After successful connection to the server, any device with the status marked as "Expired" is allowed to access Worklight Server and its status is changed to "Active". For more information about licensing, see "License Tracking report" on page 1465.

Clicking the + icon in the column shows a list of all applications that this device accessed.

Figure 218. 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 870.

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 1447.

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 1464.

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:
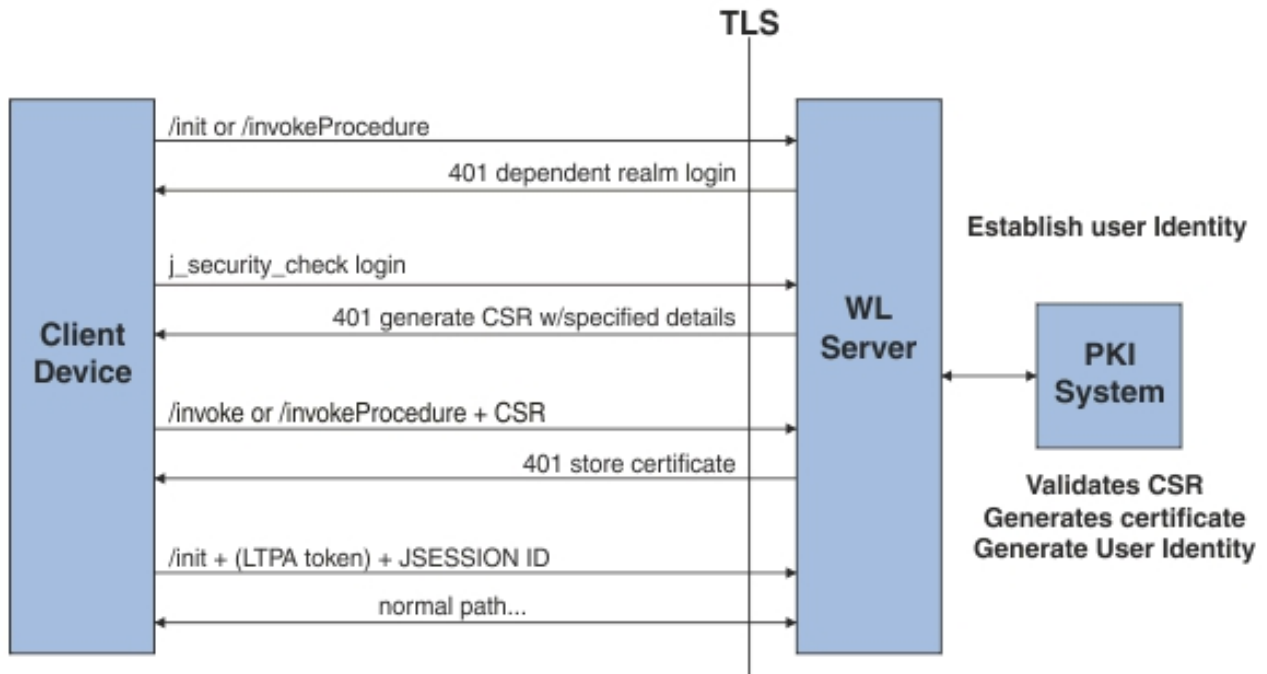
Subsequent calls from that Worklight application use this X.509 certificate to establish a secure connection over HTTPS, authenticate the user, and establish the user identity on the server. Users need to log in only once for the life of the certificate. When the certificate expires or is revoked by 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:
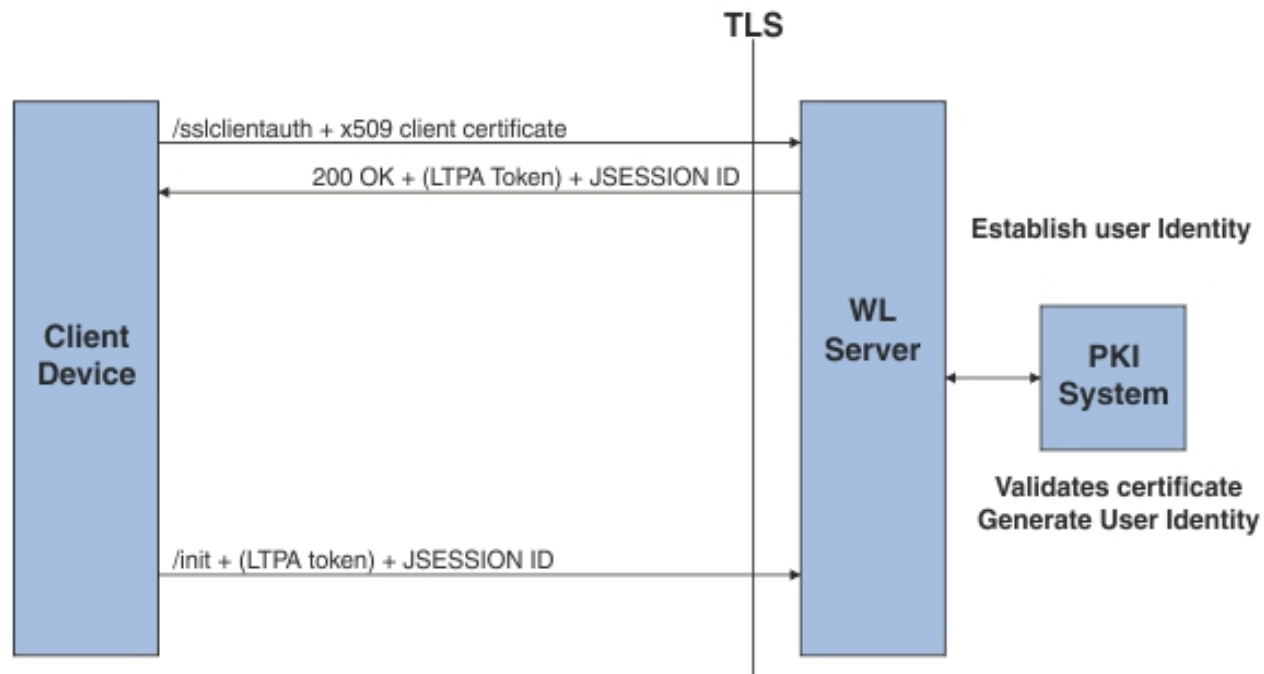
Protecting resources with user certificate authentication

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 1460.
4. Configure the server.
   a. Configure your WebSphere Application Server Liberty profile server. For more information, see "Configuring the Liberty profile" on page 1459.
   b. Configure the server for HTTPS. For more information, see "SSL configuration" on page 1451.
   c. Configure an embedded PKI or external PKI. For more information, see "PKI bridge configuration" on page 1452.
   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 1459.

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 by using untrusted certificates" on page 210.

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 9, *Authentication and security* in Chapter 4, "Tutorials and samples," on page 35.

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 1458.

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 9, *Authentication and security* in Chapter 4, "Tutorials and samples," on page 35.

If you encounter errors with certificates that are not signed by well-trusted CAs, see "Configuring SSL by using untrusted certificates" on page 210.

Certificate chains, keystore, and trust store

You must set the server certificate as the Worklight Server 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 1458.

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/
EyHLNfdr1FTDAApOERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+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/
EyHLNfdr1FTDAApOERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+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
+E2KroFKNPyXdHAL7mwpkZOOaSYxUYYwcu8IAureMWZGps196Swk1YptboIEUSd5r3j07rBZX81B
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
MSowKAYDVQQDEyFXb3JrbGlnaHQgR2FycmljmljayBCZXRhIFNpZ25pbmcgQ0EwHhcNMTMxMDA4M
DA1MzEzWhcNMTMxMDIyMDA1MzEzWjAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjA
NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZK
cf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2Qggq

```
pBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJOm2bcHRswIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQBw40DwQo5AGIgq12EDMbry
mbQUxGQ19tUAjJmW22n4n2ddiaM79oY4QmTfugbMQ9M7m4D94MCu9D1YXcG3M5OX2HDiPcA
ImOjmgScPtRNds/MraTXDcg1fhy44icuLsQkQpv4K47m4zLWZQ6PHVqP9Zor4TYqugUo0/
Jd0cAvubCmRk7RpJjFRhjBy7wgC6t4xZkamzX3pLCTVim1uggRRJ3mvePTusFlfzUEBflrCoTHe2ls/
eqkgkgr7rGiK7a0vruYqomP2XX9hv+pCZSDqi4m2319ZhQ7unXAmBSb03R6LFd+0q2OnfYWn
+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-DA25-46A7-8A79`

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 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.
- The User Certificate Authentication feature requires the server to be configured to require a valid X.509 client certificate. The feature also requires an alternate fallback authentication mechanism when a certificate does not yet exist on the client. WebSphere Application Server Liberty Profile Versions 8.5.5.0 and 8.5.5.1 allow a basic authentication, or a HTTP 401 status code, as a fallback to authenticate a user. However, a Worklight client cannot handle this configuration. If you want to protect the 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 PI10103: Support certificate authentication to fail over to a form-based login.

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 truststore 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 1452) or adapter-based ("External/adapter-based PKI bridge" on page 1454) or supply your own custom PKI bridge implementation ("Custom PKI bridge" on page 1458).

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 1452) or Adapter-Based ("External/adapter-based PKI bridge" on page 1454) 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 Worklight adapters" on page 734. 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 by using untrusted certificates" on page 210.

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.

Group support

User certificates are issued by default to a user on a specific application and device. Group support allows a certificate to be issued 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 1462.

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 or family 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 (single sign-on). This single sign-on option among a family of applications can be achieved with the Simple Data Sharing feature. The Simple Data Sharing feature allows the User Certificate Authentication feature to provision a device with a user certificate that applies to, and is used by, all applications in the same specified Worklight family.

About this task

You can configure the User Certificate Authentication feature to provision the device with a user certificate that is shared among a group of applications. This configuration allows a group of applications to authenticate with the same X.509 client certificate. This function is supported only on iOS and Android hybrid environments.

Note: The iOS `x509AccessGroup` property is deprecated since IBM Worklight Foundation V6.2.0. Use the Simple Data Sharing feature instead.

Procedure

1. Enable the Simple Data Sharing feature as explained in "Enabling the Simple Data Sharing feature for hybrid applications" on page 927.
2. Ensure that you select the user certificate authentication group support option in the application descriptor file.

☑ Enable User Client Certificate Authentication Group Support

When you use the User Certificate Authentication feature, enable this option to provision the device with a user certificate that is shared by all applications in the same family.

Troubleshooting the User Certificate Authentication feature

Find solutions to problems with the User Certificate Authentication feature.

Table 323. User Certificate Authentication troubleshooting guidelines. This table lists possible problems and actions to take to troubleshoot the User Certificate Authentication feature.

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 1460.
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. . Verify that the `debuggable` flag is set to `false` (`debuggable:false`) in the Android manifest file, 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 by using untrusted certificates" on page 210.
`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 1451. 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.

Table 323. User Certificate Authentication troubleshooting guidelines (continued). This table lists possible problems and actions to take to troubleshoot the User Certificate Authentication feature.

Problem	Actions to take
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 Foundation 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, 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 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 Foundation 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

CSV

Close

Download File

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 15. Integrating with other IBM products

IBM Worklight Foundation integrates with many other IBM products.

You can find documentation and samples for developers and administrators about how to integrate IBM Worklight Foundation with other IBM products on the developerWorks Integration page.

This series of topics complements the developerWorks Integration page with additional information about IBM Endpoint Manager for Mobile Devices, IBM WebSphere Cast Iron, IBM WebSphere DataPower, and IBM Security Access Manager.

Introduction to Worklight integration options

Developers and administrators can use integration options that are available as part of the larger IBM Worklight Foundation offering, as introduced here.

IBM Worklight Foundation provides extensible connectivity options to external resources by using the adapter technology available in IBM Worklight Foundation. IBM Worklight Foundation also provides a flexible authentication framework to support existing security requirements through the authenticator or login modules.

Figure 219 on page 1468 gives a high-level view of the topology context for an app on a device that connects to IBM Worklight Foundation. It also shows how IBM Worklight Foundation 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 Foundation
EBE	Existing back ends
I	Other Internet sources
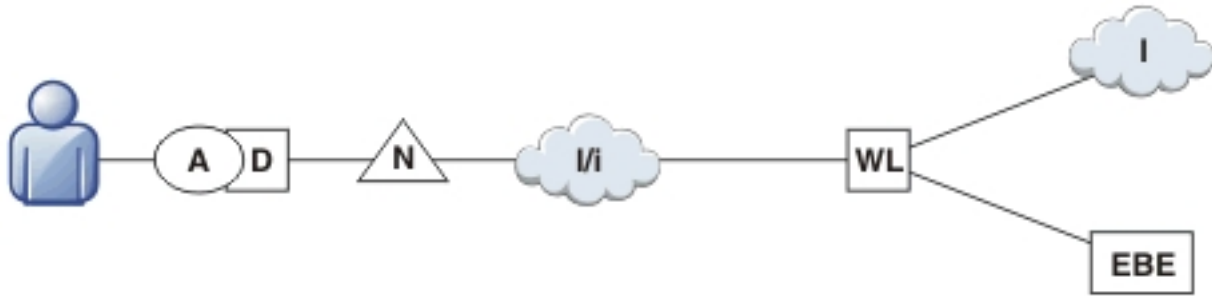
Figure 219. Overall Topology

> Figure 220 shows where these products fit within the typical Worklight topology diagram shown in Figure 219.
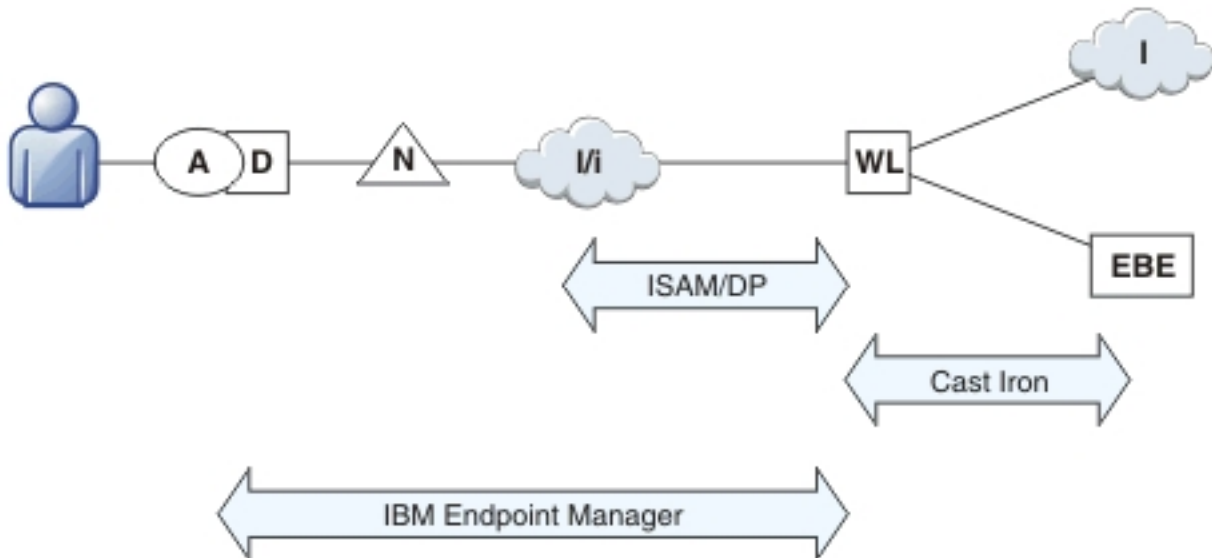
Figure 220. Integration Points

Integration with Cast Iron

> An overview of the use of IBM WebSphere Cast Iron to enable enterprise connectivity within a Worklight environment.
>
> There are four adapters supported as part of IBM Worklight Foundation:
> - 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 221 shows how the topology in Figure 1 in *Introduction to IBM Worklight integration options* changes to reflect the use of Cast Iron, with the Worklight Cast Iron adapter represented by the thicker line between IBM Worklight Foundation and Cast Iron.

Figure 221. 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 4, "Tutorials and samples," on page 35.

Integration with reverse proxy

An overview of the use of a reverse proxy to enable enterprise connectivity within a Worklight environment.

Reverse proxies typically front Worklight run times as part of the deployment shown in Figure 222, and follow the gateway pattern.
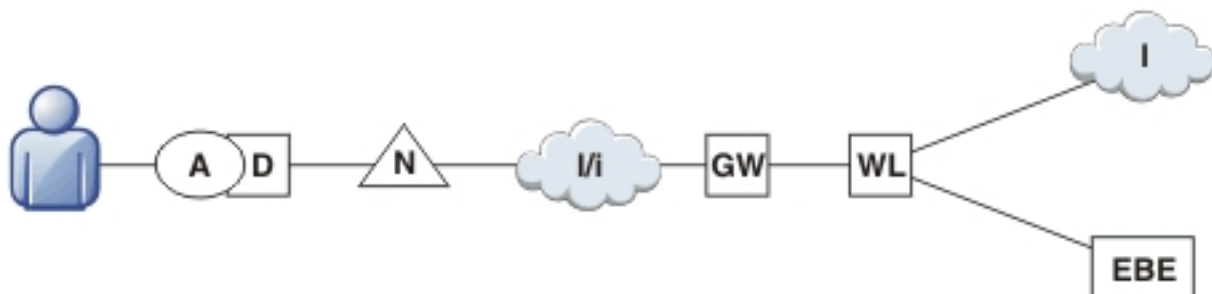
Figure 222. Integration with reverse proxy

The gateway icon (GW) represents a reverse proxy such as WebSphere DataPower, or IBM Security Access Manager. In addition to protecting 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 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 Foundation.

If authentication is terminated at the gateway, IBM Worklight Foundation 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 Foundation 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 223.
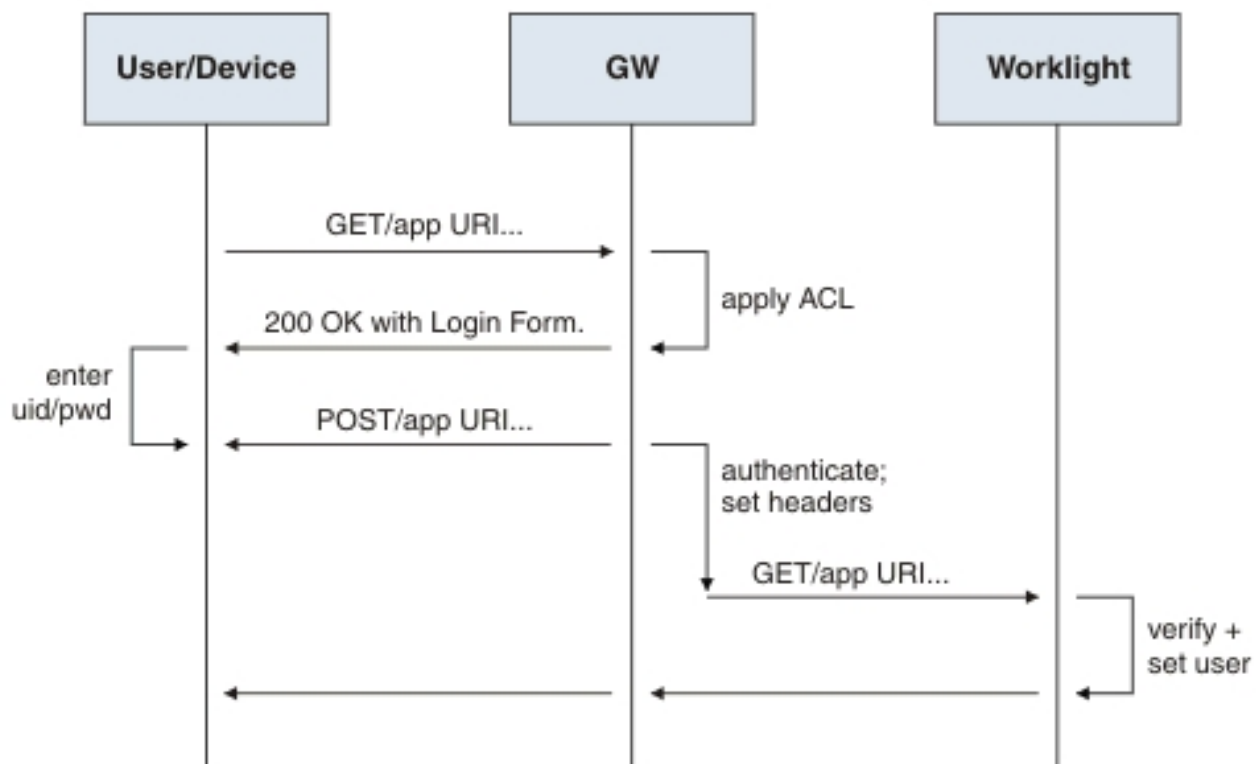
Figure 223. Authentication flow

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 Foundation through a reverse proxy.

- On successful authentication, the gateway forwards a custom HTTP header with the user name or ID to IBM Worklight Foundation.
- IBM Worklight Foundation 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 Foundation through a reverse proxy.

- On successful authentication, the gateway forwards an LTPA token (in the form of an HTTP cookie) toIBM Worklight Foundation
- IBM Worklight Foundation on WebSphere Application Server is configured to use `WebSphereFormBasedAuthenticator` and `WebSphereLoginModule`.

Integration with IBM Endpoint Manager

An overview of the features and architecture of IBM Endpoint Manager.

IBM Endpoint Manager for Mobile Devices overview

An overview of the features and architecture of IBM Endpoint Manager for Mobile Devices.

IBM Worklight Foundation 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.
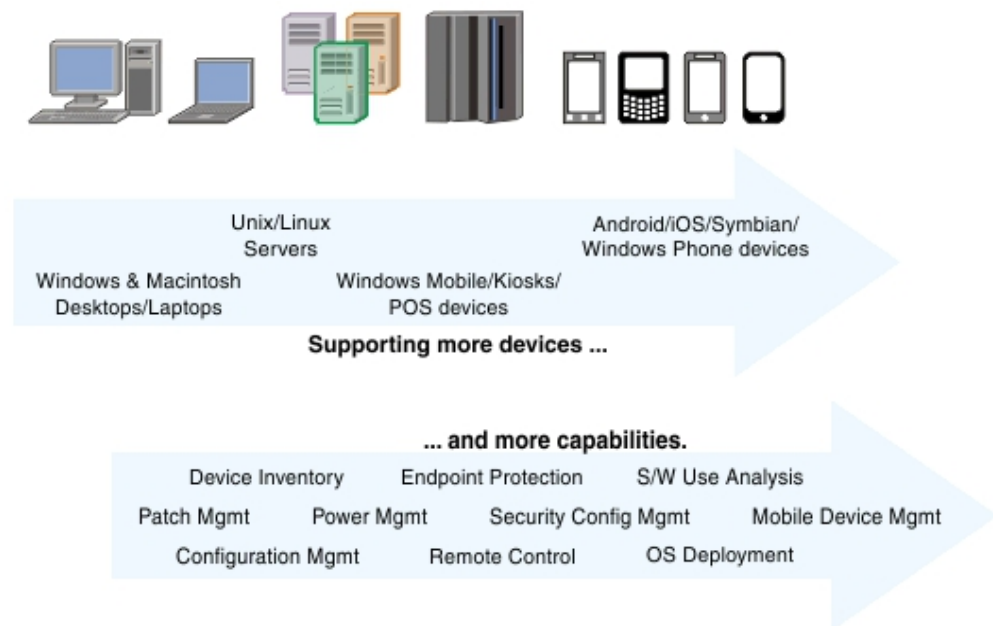
The most interesting capabilities that are provided by IBM Endpoint Manager for Mobile Devices from a security standpoint are delivered in the following areas:

- 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 it is isolated from the rest of the device. This isolation 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 indicated in the previous section.
- PIM is available through NitroDesk TouchDown.

The following diagram shows an architectural overview of a production-level, agent-based, MDM API-based implementation with IBM Endpoint Manager for Mobile Devices.
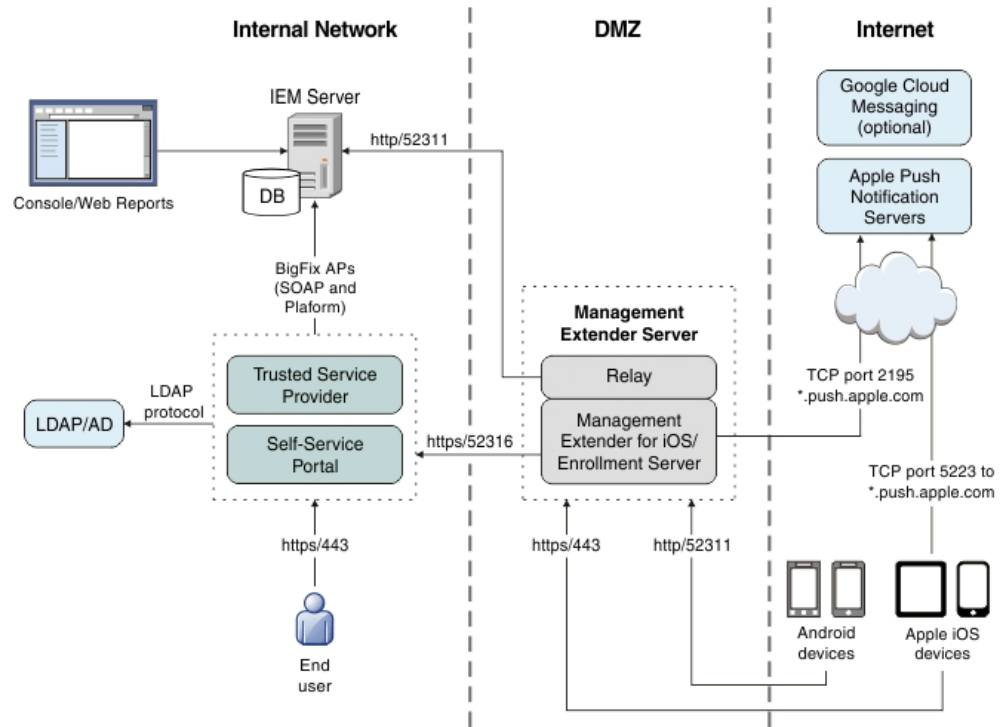
Managing end points with IBM Endpoint Manager

An overview of the use of IBM Endpoint Manager for Mobile Devices to manage devices within a Worklight environment.

IBM Worklight Foundation 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 Foundation and IBM Endpoint Manager for Mobile Devices, as shown in Figure 224 on page 1474.
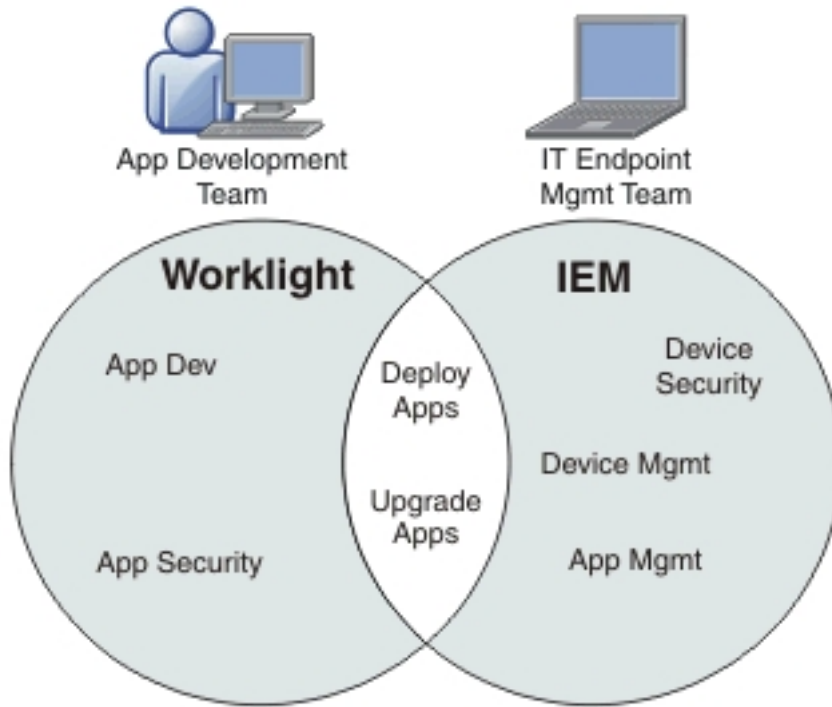
Figure 224. IBM Worklight Foundation and IBM Endpoint Manager management capabilities

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.

Integration with IBM Tealeaf

An overview of the use of IBM Tealeaf.

IBM Tealeaf CX Mobile helps customers apply the power of Tealeaf's powerful customer experience management solutions to their mobile websites, native applications, and hybrid applications, including support for HTML5. IBM Tealeaf gives customers visibility where they do not have it today, helping to deliver winning mobile services. IBM Tealeaf CX Mobile is an add-on to the Tealeaf CX platform.

IBM Tealeaf CX Mobile helps customers to:

Optimize customer experiences of their mobile channel.
- Gain insights into how all users interact with mobile web, HTML5-based sites, hybrid applications, or native applications (such as iOS and Android).
- Improve mobile customer conversions and grow the mobile revenue channel along with customer web channels.

Gain complete mobile visibility.
- Capture user information on mobile websites, HTML5-based sites, native applications, and hybrid applications, including both network and client interactions.
- Understand why users struggle with enhanced mobile web session replay by viewing what customers saw on their mobile devices and the specific actions they took, including device rotation, scroll, swipe, and other touchscreen actions.
- Analyze data from mobile customer sessions in real time against a set of user-defined rules or events to gain immediate awareness and insight into the issues that cause customers to struggle.
- Provide HTML and HTML5 support to automatically determine the highest impact customer-facing issues.

Build and manage an early warning system.
- Detect mobile user problems and provide proactive awareness into mobile application failures, usability issues, or other obstacles that lead to failed transactions, abandonment, poor application store ranking, and negative feedback.
- Use powerful visitor and device data to further understand mobile users' usage statistics and patterns.

Quantify revenue impact and segmentation
- Deliver ad hoc mobile segmentation with real-time drag-and-drop analysis by specific mobile user behaviors or device attributes.
- Create segmentation analysis that is based on mobile attributes, such as device manufacturer, operating system, browser type, screen resolution, and others.
- Share and distribute reports to management and key stakeholders.

Quickly find and isolate problems within mobile customer sessions
- Find and retrieve specific mobile sessions by searching mobile-specific attributes (phone model, screen size, browser version, and others), HTTP parameters, free-text strings, or events that are observed by IBM Tealeaf CX platform.

- Gain powerful ad hoc discovery and segmentation without predefining tags or beacons.

Note: IBM Tealeaf is provided as a set of libraries. You must take steps on both your client and your server to use it.

IBM Tealeaf client-side integration

For your convenience, Worklight Studio can be used to place IBM Tealeaf client SDK libraries and configuration files into the appropriate Worklight project file system locations for supported environments so that the IBM Tealeaf client SDK API is available in development. After the libraries and configuration files are placed, the IBM Tealeaf client SDK API is on your class path (Java), available in XCode (Objective-C), and the TLT global variable is available in your JavaScript code.

About this task

Integrating the IBM Tealeaf client SDK into your application is only meaningful when you also have an IBM Tealeaf CX server to which the clients send their collected data. Use the Worklight Studio `application-descriptor.xml` editor to manage the optional feature.

Procedure

1. Find and double-click the `application-descriptor.xml` file to open the Application Descriptor editor view.
2. Find and click **Optional Features**.
3. Click **Add...**.
4. Click **IBM Tealeaf SDK**.
5. Click **OK**.

Results

You have placed the IBM Tealeaf SDK client artifacts in your project.

Note:
- Selecting the IBM Tealeaf SDK for inclusion in your application in Worklight Studio does not overwrite existing IBM Tealeaf artifacts.
- Removing the IBM Tealeaf SDK from inclusion in your application by removing the optional feature item removes existing IBM Tealeaf artifacts from your project, including any you placed manually.

Knowing this information, you can place specific versions of the IBM Tealeaf SDK artifacts manually, and avoid managing the placement of IBM Tealeaf artifacts in your project in the application descriptor editor.

For more information about how to properly edit the properties file (Android), the `pList` file (iOS), or the configuration object (JavaScript), and how to exercise the IBM Tealeaf CX Mobile libraries' API, see Tealeaf CX Mobile.

Example

The following figure shows the Application Descriptor editor view with the steps highlighted:
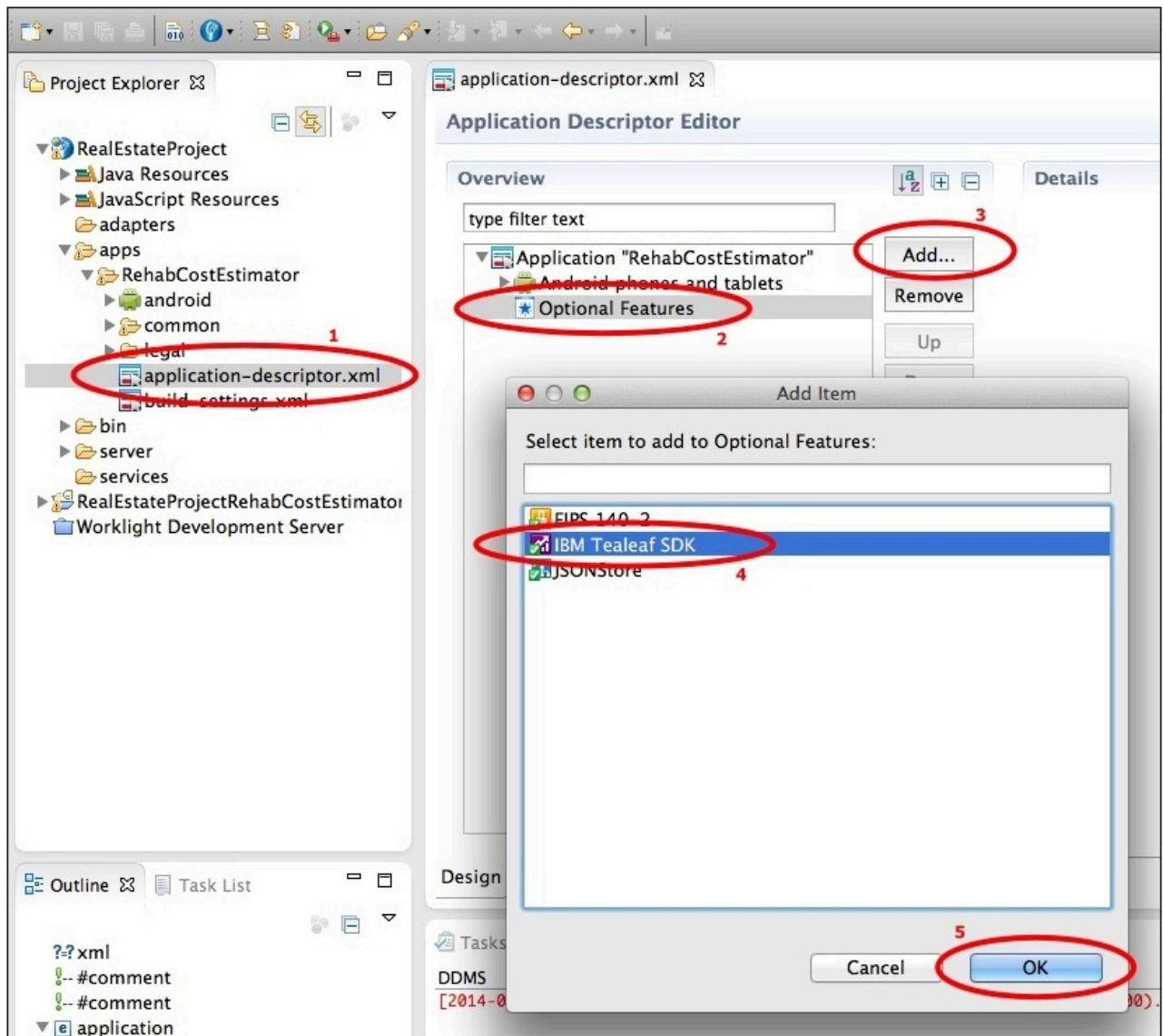
IBM Tealeaf server-side integration

You must install the IBM Tealeaf Mobile CX server to aggregate data that is collected in client applications by the IBM Tealeaf client SDK.

For more information about installation, configuration, and usage, see Tealeaf CX Mobile.

Integration with IBM Trusteer

An overview of the use of IBM Trusteer.

Overview

IBM Worklight Foundation supports full integration with the IBM Trusteer Mobile SDK, for Android and iOS applications only.

The SDK collects multiple mobile device risk factors and provides them to the mobile app, so enabling organizations to restrict mobile app functionality, based on risk levels. A Worklight application that includes the Trusteer SDK can detect malware or determine whether the device on which it is running is jailbroken or rooted, so protecting access to specific resources or procedures of the application. For example, you could prevent a malware-ridden device from logging into your banking app, and prevent rooted devices from using the "transfer funds" feature.

After you have installed the Trusteer Mobile SDK, you must configure Worklight Server to support it:

- "Worklight security overview" on page 1201 and "Worklight security configuration" on page 1202 provide an overview of general Worklight security configuration.
- "Configuring the Worklight Server for Trusteer" on page 1207 provides steps for configuring Worklight Server to support Trusteer.

Obtaining Trusteer Software

Depending on your development environment, it is recommended you obtain the Trusteer Mobile SDK either as a Worklight component, or as a zipped archive:

- For hybrid Android applications, use the Trusteer Worklight component as is.
- For hybrid iOS applications, use either the Trusteer Worklight component, or Trusteer zipped archive.
- For pure native iOS or Android applications, use a Trusteer zipped archive.

For more information about obtaining Trusteer software and licenses, see the developerWorks Integration page.

Integrating IBM Trusteer for iOS

You might want to integrate Trusteer with IBM Worklight for iOS apps.

Before you begin

- If you are using a Trusteer zipped archive:
  - Make sure that the you have the following files:
    - Trusteer Mobile iOS library (`libtas_full.a`).
    - A Worklight-compatible Trusteer license file (`tas.license`)
    - A Trusteer configuration package (`default_conf.rpkg`)
    - A Trusteer Application Security Manifest (`manifest.rpkg`)

      Note: You might need to generate the manifest manually. For more information, see the Trusteer documentation.
    If any of these items is missing, consult your local IBM representative.
  - In your file system, create a `tas` folder and place in it the files listed previously.
  - Now continue from step 1.
- If you are using a Trusteer Worklight component, first follow the instructions in "Adding application components to Worklight projects" on page 701, then continue from step 1, below.

Procedure

1. In your Xcode project, drag the folder onto your project navigator.
2. Select the check box **Copy items into destination group's folder (if needed)**.
3. Select the option **Create folder references for any added folders**.
4. Make sure that your target is selected, then click **Finish**.
5. Click the project name at the top of the tree in the **Project Navigator**, then click **Build Phases**.
6. To link your project with the Trusteer library, drag the `libtas_full.a` file from the **Project Navigator** to the **Link Binary With Libraries** list.

   Note: To avoid possible link issues, arrange the items in the list so that `libWorklightStaticLibProject.a` appears at the top and `libtas_full.a` appears next.
7. In **Build Settings** > **Linking** > **Other Linker Flags**, add: `-force_load "$(SRCROOT)/tas/libtas_full.a"`.
8. In **Build Settings** > **Linking** > **Dead Code Stripping**, select **NO**.
9. In the Xcode **Project Navigator**, drag `tas.license` into the **Resources** group. In the dialog that appears, click **Finish**.
10. Open the `tas.license` file and check that the values for `vendorId`, `clientId` and `clientKey` match the licensing information that was provided by Trusteer.

Integrating IBM Trusteer for Android using a Worklight Component

You might want to integrate Trusteer with IBM Worklight for Android apps using a Worklight Component.

Procedure

1. Make sure that you have obtained a Trusteer Worklight Component.
2. Add the Trusteer Worklight Component to your Worklight project, by following the procedures in "Adding application components to Worklight projects" on page 701.
3. In the **Project Navigator**, under your application, navigate to **nativeResources** > **assets**.
4. Open the `tas.license` file and check that the values for `vendorId`, `clientId` and `clientKey` match the licensing information that was provided by Trusteer.

Integrating IBM Trusteer for Android from a zipped archive

You might want to integrate Trusteer with IBM Worklight for Android apps using a zipped archive.

Procedure

1. Make sure that Trusteer has provided you with the following:
   a. Trusteer Mobile Android native libraries (`libtaz_full.so` and `libgnustl_shared.so`).
   b. Trusteer Mobile Android Java Libraries (`taz.jar` and `reflectutils.jar`).
   c. Worklight-compatible Trusteer license file (`tas.license`).
   d. Trusteer configuration package (`default_conf.rpkg`).

   If any of these items is missing, consult your local IBM representative.
2. In your file system, create two folders: `libs/armeabi` and `libs/armeabi-v7a`.

3. Copy the Trusteer Mobile Android native libraries to the two folders.
4. Copy the Trusteer Mobile Android Java Libraries to the `libs` folder.
5. Copy the Trusteer configuration package (`default_conf.rpkg`) to the `Assets` folder.
6. Copy the Worklight-compatible Trusteer license file (`tas.license`) to the `Assets` folder.
7. Add the following permissions to the application manifest file (`androidManifest.xml`):

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

8. Optional: Add the following permission to the application manifest file (`androidManifest.xml`):

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

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 Foundation 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 Foundation 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 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

      http://www.ibm.com/support/docview.wss?uid=swg24034222

Chapter 16. Glossary

This glossary provides terms and definitions for the IBM Worklight Foundation V6.2.0 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

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 a Worklight application. Adapters connect to enterprise applications, deliver data to and from mobile applications, and perform any necessary server-side logic on sent data.

Administration Services
An application that hosts the REST services and administration tasks. The Administration Services application is packaged in its own WAR file.

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
A 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 Center Installer must be present on a device in order to install applications from your private application repository.

application descriptor file
A metadata file that defines various aspects of an application.

application programming interface (API)
An interface that allows an application program that is written in a high-level language to use specific data or functions of the operating system or another program.

authentication
A security service that provides proof that a user of a computer system is genuinely who that person claims to be. Common mechanisms for implementing this service are passwords and digital signatures.

authenticator
1. A server-side component that issues a sequence of challenges on the server side and responds on the client side. See also challenge handler.
2. In the Kerberos protocol, a string of data that is generated by the client and sent with a ticket that is used by the server to certify the identity of the client.

B

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).

broadcast notification
A notification that is targeted to all of the users of a specific Worklight application. See also tag-based notification.

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 collection of complete systems that work together to provide a single, unified computing capability.

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.

farm node
>   A networked server that is housed in a server farm.

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

IBM Worklight administration database

The database of the Worklight Console and of the Administration Services. The database tables define elements such as applications, adapters, projects with their descriptions and orders of magnitude.

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.

J

Java Management Extensions (JMX)

A means of doing management of and through Java technology. JMX is a universal, open extension of the Java programming language for management that can be deployed across all industries, wherever management is needed.

JMX See Java Management Extensions.

K

key

1. A cryptographic mathematical value that is used to digitally sign, verify, encrypt, or decrypt a message. See also private key, public key.

2. One or more characters within an item of data that are used to uniquely identify a record and establish its order with respect to other records.

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 Foundation 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

Managed Bean (MBean)

In the Java Management Extensions (JMX) specification, the Java objects that implement resources and their instrumentation.

MBean

See Managed Bean.

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 contains the configurations for the Worklight runtime environment and is deployed on an application server.

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 farm
A group of networked servers.

server-side authentication component
See authenticator.

service
A program that performs a primary function within a server or related software.

session
A logical or virtual connection between two stations, software programs, or devices on a network that allows the two elements to communicate and exchange data for the duration of the session.

shell   A component that provides custom native capabilities and security features for applications.

sign   To attach a unique electronic signature, derived from the sender's user ID, to a document or field when a document is mailed. Signing mail ensures that if an unauthorized user creates a new copy of a user's ID, the unauthorized user cannot forge signatures with it. In addition, the signature verifies that no one has tampered with the data while the message was in transit.

simulator
An environment for staging code that is written for a different platform. Simulators are used to develop and test code in the same IDE, but then deploy that code to its specific platform. For example, one can develop code for a BlackBerry device on a computer, then test it using a simulator on that computer.

skin   An element of a graphical user interface that can be changed to alter the appearance of the interface without affecting its functionality.

slide   To move a slider interface item horizontally on a touchscreen. Typically, apps use slide gestures to lock and unlock phones, or toggle options.

subelement
In UN/EDIFACT EDI standards, an EDI data element that is part of an EDI composite data element. For example, an EDI data element and its qualifier are subelements of an EDI composite data element.

subscription
A record that contains the information that a subscriber passes to a local broker or server to describe the publications that it wants to receive.

syntax   The rules for the construction of a command or statement.

system message
An automated message on a mobile device that provides operational status or alerts, for example if connections are successful or not.

T

tag-based notification
A notification that is targeted to devices that are subscribed for a specific tag. Tags are used to represent topics that are of interest to a user. See also broadcast notification.

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.

U

Unstructured Supplementary Service Data (USSD)
A communication technology that is used by GSM cellular telephones to send text messages between a mobile phone and an application program in the network. USSD establishes a real-time session between the mobile phone and the application that handles the service.

USSD See Unstructured Supplementary Service Data.

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 Worklight runtime environments that are deployed in Worklight Server, and to collect and analyze user statistics.

Worklight runtime environment

A mobile-optimized server-side component that runs the server side of your mobile applications (back-end integration, version management, security, unified push notification). Each runtime environment is packaged as a web application (WAR file).

Worklight Server

A Worklight component that handles security, back-end connections, push notifications, mobile application management, and analytics. The Worklight Server is a collection of apps that run on an application server and acts as a runtime container for Worklight runtime environments.

Worklight Studio

A 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 17. Notices

This information was developed for products and services offered in the US. This material might be available from IBM in other languages. However, you may be required to own a copy of the product or product version in that language in order to access it.

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, MD-NC119
Armonk, NY 10504-1785
US

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.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

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 jurisdictions 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 websites are provided for convenience only and do not in any manner serve as an endorsement of those

websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you provide 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 Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
US

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.

The performance data and client examples cited are presented for illustrative purposes only. Actual performance results may vary depending on specific configurations and operating conditions.

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.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to actual people or business enterprises is entirely coincidental.

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. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work must include a © (your company name) (year).
Portions of this code are derived from IBM Corp. Sample Programs.
© Copyright IBM Corp. _enter the year or years_.

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 "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.

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.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Node.js is a trademark of Joyent, Inc. and is used with its permission. This documentation is not formally endorsed by or affiliated with Joyent.

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.

Terms and conditions for product documentation

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 Online Privacy Statement

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.

Chapter 18. Support and comments

For the entire 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 19. Terms and conditions for product documentation

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

Special characters

<adapter>
 element of adapter XML file   739
<authentication>
 element of the HTTP adapter   745
<connectionPolicy>
 element of adapter XML file   741
 element of the Cast Iron adapter   749
 element of the HTTP adapter   743
 element of the JMS adapter   750
 element of the SAP adapter   752
 element of the SQL adapter   747
<connectivity>
 element of adapter XML file   741
<jmsConnection>
 element of the JMS adapter   751
<loadConstraints>
 element of adapter XML file   741
<namingConnection>
 element of the JMS adapter   750
<procedure>
 element of adapter XML file   742
<proxy>
 element of the HTTP adapter   746
 316

A

Access Control List
 Application Center   274, 276
access for users and groups
 Application Center   274, 276, 277
accessibility   956
accuracy   539
ACL
 Application Center   274, 276
ACL management for Application Center
 with LDAP
 WebSphere Application Server
  V8   277
acquisition policy
 setting   964
actions and data objects   477
 overview   477
 sending   477
adapter   756
adapter configuration files
 exporting   1200
adapter framework   734
adapter invocation   761
adapter procedures
 implementing   762
adapter validation   768
adapter XML file   738
 <adapter> element   739
 <connectionPolicy> element   741
 <connectivity> element   741
 <loadConstraints> element   741
 <procedure> element   742
adapter-based authenticator   892

adapters   19, 756
 See also HTTP adapters
 administering in console   1198
 anatomy   734
 and SSL configuration   209
 backend responses   764
 benefits   734
 building
  Ant task   1190, 1192
 Cast Iron
  See Cast Iron adapters
 composition   734
 configuring   754
 creating   754
 deleting   1201
 deploying
  Ant task   1190, 1194
  from the console   1200
  from Worklight Studio   776
 deploying between
  environments   1101, 1102
 HTTP
  See HTTP adapters
 JMS
  See JMS adapters
 modifying   1200
 overview   734
 replacing   1200
 SQL
  See SQL adapters
adding
 desktop environment   438
 mobile environment   438
 to an
  worklight application   438
 web environment   438
adding custom splash images   423
administering
 applications   1247
 apps and adapters
  in Worklight Console   1198
administration   104, 105, 106, 131, 1416,
 1417, 1418
administration databases   104
Administration Services
 installing during an upgrade   382
 preparing the installation   359
Adobe AIR
 in application descriptors   430
Adobe AIR applications
 signing   597
Adobe AIR tools
 installing   60
AIR
 See also Adobe AIR
 See Adobe AIR   430
AIR applications
 signing   597
AMD   506

analytics   6, 232, 237, 1395, 1397, 1401,
 1402, 1403, 1404, 1408, 1415, 1416, 1417,
 1418, 1419
 cluster deployment   1414
 configuring   237
 installing   233, 234
 production cluster setup   1412
 securing   1401
and application authenticity   866
android   623
Android   538
 application authenticity   866
 configuring SSL with untrusted
  certificates   210
Android 6.0 support   16
Android apps   465
Android automatic backup   16
Android examples   839
Android location services   16
Android marshmallow support   16
Android native   929
Android Studio   465
Android tools
 installing   61
animating transitions
 from and to Java page   595
 from Objective-C page to web
  view   593
 from web view to Objective-C
  page   593
ANT Grunt   640
Ant tasks
 application servers   1156
 building adapters   1190, 1192
 building applications   1190, 1192
 building projects   1104
 configuring application servers   117,
  1110, 1141, 1146, 1156, 1162
  WebSphere Application Server
   Network Deployment   1162
 configuring database   113
 configuring databases   1110, 1114,
  1115
 deploying adapters   1190, 1194
 deploying applications   1190, 1194
 deploying projects   1190
 for product upgrades   361
 for the installation of server
  farms   165
 for Worklight installation   120
 reference   1156
 sample configuration files   1156
anti   873
Apache   251, 1131
Apache Tomcat   153
Apache Tomcat server
 manual configuration   142, 258, 1140
API   19, 930
API reference   1001
APIs
 Apache Cordova globalization   943