IBM MobileFirst Platform Foundation for iOS V7.1.0

IBM

Note

Before you use this information and the product it supports, read the information in “Notices” on page A-1.

IBM MobileFirst Platform Foundation for iOS V7.1.0

This edition applies to version V7.1.0 of IBM MobileFirst Platform Foundation for iOS and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition was updated last on 24 May 2017.

This PDF document is made available for convenience and on an "as is" basis only. The master and controlling document can be found in Knowledge Center at http://ibm.biz/knowctr#SSHSCD_7.1.0/wl_welcome.html. This PDF document may contain uncontrollable formatting errors or differences from the master version in Knowledge Center.

Contents

Monitoring and mobile operations 13-1

IBM MobileFirst Platform Foundation for iOS V7.1.0 documentation

Welcome to the IBM MobileFirst™ Platform Foundation for iOS V7.1.0 documentation, where you can find information about how to install, maintain, and use the IBM MobileFirst Platform Foundation for iOS.

Getting started

"Product overview" on page 2-1
IBM MobileFirst Platform Foundation for iOS is an integrated platform that helps you extend your business to mobile devices.

"Notices" on page A-1

"Release notes" on page 3-1
You can identify the latest information about this product release and all its fix packs.

"Tutorials, samples, and additional resources" on page 5-1
Tutorials and samples help you get started with and learn about IBM MobileFirst Platform Foundation for iOS. Use them to evaluate what the product can do for you.

"Installation overview" on page 6-1
IBM MobileFirst Platform Foundation for iOS provides the following installable components: MobileFirst Platform Command Line Interface for iOS and MobileFirst Server. This section gives an overview of the installation process.

"Configuring MobileFirst Server" on page 6-136
Consider your backup and recovery policy, optimize your MobileFirst Server configuration, and apply access restrictions and security options.

"System requirements" on page 2-9
System requirements for IBM MobileFirst Platform Foundation for iOS include operating systems, SDKs, and other software.

Common tasks

"Developing MobileFirst applications" on page 8-1
You use IBM MobileFirst Platform Command Line Interface for iOS, the MobileFirst client, and the server-side API to develop iOS applications.

"Deploying MobileFirst projects" on page 11-1

"Administering MobileFirst applications" on page 12-1
Run and maintain MobileFirst applications in production.

"Application Center" on page 12-65
Learn about the Application Center: what it is for, the different components and features, and how to use the console and the client.

Troubleshooting and support

"Troubleshooting" on page 4-1
You can find advice on how to troubleshoot problems, and more information about known limitations and technotes (Troubleshooting).

"Known issues" on page 3-16
You can identify the latest known issues and their resolutions, for this product release and all its fix packs, by browsing this dynamic list of documents.

"Accessibility" on page 2-11
Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with IBM MobileFirst Platform Foundation for iOS.

IBM Software Support home page

More information

Latest version of this PDF file

Online knowledge center for this documentation

Mobile Application Developer skills

Product overview

IBM MobileFirst Platform Foundation for iOS is an integrated platform that helps you extend your business to mobile devices.

IBM MobileFirst Platform Foundation for iOS 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.

With IBM MobileFirst Platform Foundation for iOS, your organization can efficiently develop, connect, run, and manage rich mobile applications (apps) that can access the full capabilities of your target mobile devices. IBM MobileFirst Platform Foundation for iOS can help reduce time-to-market, cost, and complexity of development, and enables an optimized customer and employee user experience across multiple environments.

As part of this comprehensive mobile solution, IBM MobileFirst Platform Foundation for iOS can be integrated with application lifecycle, security, management, and analytics capabilities to help you address the unique mobile needs of your business.

Introduction to mobile application development

With IBM MobileFirst Platform Foundation for iOS, you can develop mobile applications by using native development.

IBM MobileFirst Platform Foundation for iOS provides capabilities to help you respond to the fast-paced development of mobile devices.

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.

Product main capabilities

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

Development

IBM MobileFirst Platform Foundation for iOS provides a framework that enables the development, integration, and management of secure mobile applications (apps). IBM MobileFirst Platform Foundation for iOS does not introduce a proprietary programming language or model that users must learn.

You can write native code (Objective-C and Swift). IBM MobileFirst Platform Foundation for iOS provides an SDK that includes libraries that you can access from native code.

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 MobileFirst Platform Foundation for iOS can act as this middleware solution and make communication with back-end services easier.

Push notifications

With push notifications, enterprise applications can send information to mobile devices, even when the application is not being used. IBM MobileFirst Platform Foundation for iOS includes a unified notification framework which provides a consistent mechanism for such push notifications.

Offline mode

In terms of connectivity, mobile applications can operate offline, online, or in a mixed mode. IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS can determine whether a more recent version of the application is available, and if so, give information to the user about it. 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS, 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. You can also prevent mobile applications from accessing sensitive information.

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 MobileFirst Platform Foundation for iOS includes a scalable operational analytics platform accessible in the MobileFirst Operations 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 MobileFirst Platform Foundation for iOS includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM MobileFirst Platform Foundation for iOS applications and servers, and for monitoring server health.

Application publishing

IBM MobileFirst Platform Foundation for iOS 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 , 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).

Product components

IBM MobileFirst Platform Foundation for iOS consists of the following components: MobileFirst Platform Command Line Interface for iOS, MobileFirst Server, client-side runtime components, MobileFirst Operations Console, IBM MobileFirst Platform Cloudant® Data Layer Local Edition, Application Center, and IBM Connect.

Component overview

The following figure shows the components of IBM MobileFirst Platform Foundation for iOS:

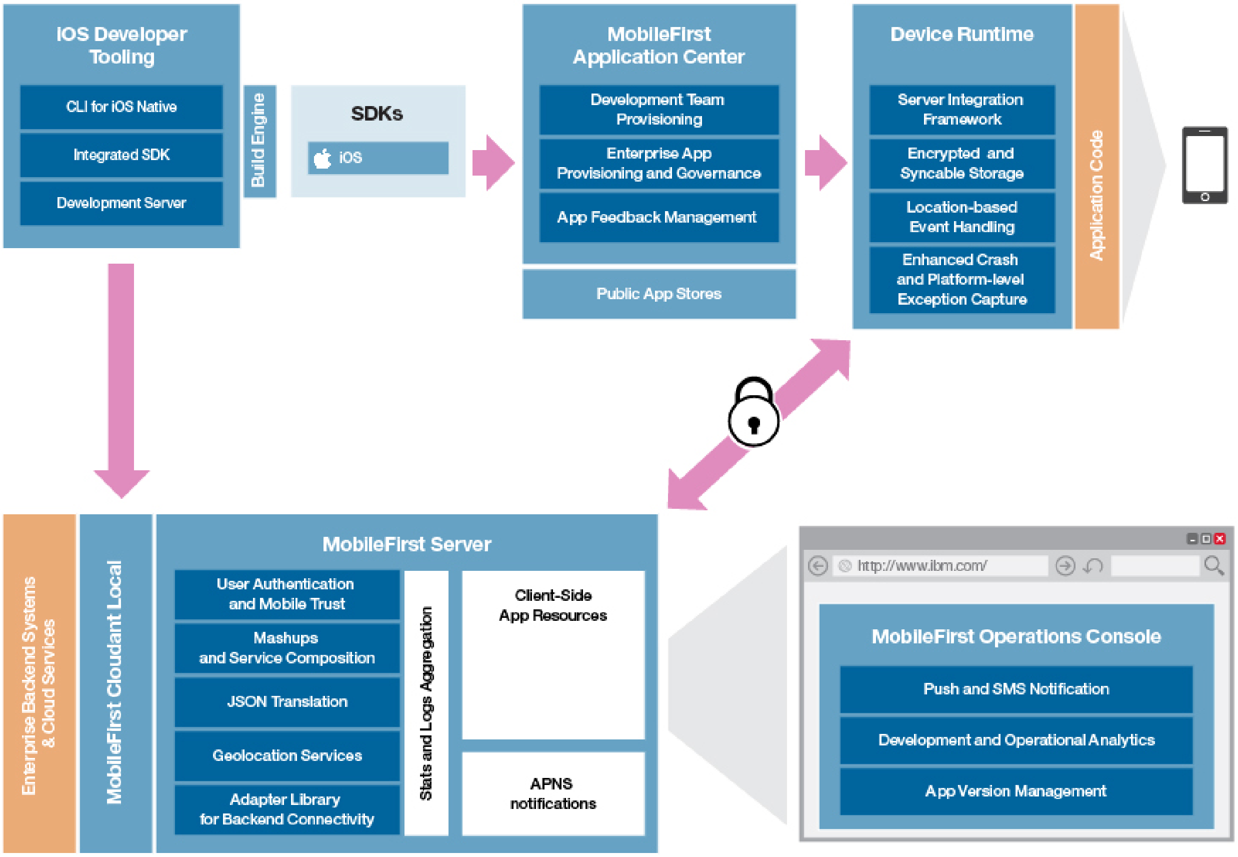

iOS Developer Tooling
CLI for iOS Native
Integrated SDK
Development Server
Build Engine
SDKs
iOS
MobileFirst Application Center
Development Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores
Device Runtime
Server Integration Framework
Encrypted and Syncable Storage
Location-based Event Handling
Enhanced Crash and Platform-level Exception Capture
Application Code
Enterprise Backend Systems & Cloud Services
MobileFirst Cloudant Local
MobileFirst Server
User Authentication and Mobile Trust
Mashups and Service Composition
JSON Translation
Geolocation Services
Adapter Library for Backend Connectivity
Stats and Logs Aggregation
Client-Side App Resources
APNS notifications
http://www.ibm.com/
MobileFirst Operations Console
Push and SMS Notification
Development and Operational Analytics
App Version Management

Figure 2-1. IBM MobileFirst Platform Foundation for iOS architecture

MobileFirst Server

The MobileFirst Server is a runtime container for the mobile applications that you develop by using MobileFirst tooling. It is not an application server in the Java™ Platform, Enterprise Edition (Java EE) sense. It acts as a container for IBM MobileFirst Platform Foundation for iOS application packages, and is in fact a collection of web applications. Optionally, those web applications are packaged as EAR (enterprise archive) files, which run on top of traditional application servers.

MobileFirst Server integrates into your enterprise environment and uses 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 MobileFirst 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 MobileFirst tools.
- 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 your organization.

MobileFirst Server environments can also be built as Docker images that you can deploy and run on the cloud using IBM Containers for Bluemix®.

IBM MobileFirst Platform Cloudant Data Layer Local Edition

MobileFirst Platform Local is an advanced NoSQL database, which can handle a wide variety of data types, such as JSON, full-text, and geospatial data.

As a JSON document store, MobileFirst Platform Local is ideal for managing multi-structured or unstructured data. With advanced indexing, you can enrich applications with location-based, geospatial services, full-text search, and real-time analytics.

MobileFirst Cloudant extensions include an SDK and proxy.

The SDK extends the Cloudant APIs with support for native language objects for iOS and Android developers, offline access, and online remote access. For more information, see "Storing mobile data in Cloudant" on page 8-151.

Authentication with OAuth is available through integration with the MobileFirst Data Proxy. For more information, see "Installing the MobileFirst Data Proxy" on page 6-200.

Client-side runtime components

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Server by defining a predefined interface for apps to access native device functions.

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 MobileFirst Server

- Mobile client functionality: access to device APIs and push notification registration

The version of Apache Cordova that is included in V7.1.0 consists of the following components.

Note: Apache Cordova-related tools are applicable only to MobileFirst Platform Command Line Interface for iOS (CLI), not to MobileFirst Studio.

Platforms

- cordova-android: 3.6.4
- cordova-blackberry10: 3.6.3
- cordova-ios: 3.7.0
- cordova-windows: 3.7.1
- cordova-wp8: 3.7.0

Plug-ins

- org.apache.cordova.battery-status: 0.2.12
- org.apache.cordova.camera: 0.3.4
- org.apache.cordova.console: 0.2.12
- org.apache.cordova.contacts: 0.2.15
- org.apache.cordova.device-motion: 0.2.11
- org.apache.cordova.device-orientation: 0.3.10
- org.apache.cordova.device: 0.2.13
- org.apache.cordova.dialogs: 0.2.11
- org.apache.cordova.file: 1.3.2
- org.apache.cordova.file-transfer: 0.4.8
- org.apache.cordova.geolocation: 0.3.11
- org.apache.cordova.globalization: 0.3.3
- org.apache.cordova.inappbrowser: 0.5.4
- org.apache.cordova.media-capture: 0.3.5
- org.apache.cordova.media: 0.2.15
- org.apache.cordova.network-information: 0.2.14
- org.apache.cordova.splashscreen: 0.3.5
- org.apache.cordova.statusbar: 0.1.9
- org.apache.cordova.vibration: 0.3.12

Tools

- cordova-cli: 5.0.0
- ios-sim: 3.1.1
- ios-deploy: 1.5.0

MobileFirst Operations Console

The MobileFirst Operations Console is used for the control and management of the mobile applications.

You can use the MobileFirst Operations 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 MobileFirst 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, preconfigured 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 your organization.

IBM MobileFirst Platform Operational Analytics

IBM MobileFirst Platform Foundation for iOS includes a scalable operational analytics feature that is accessible from the MobileFirst Operations 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:

- Crash events of an application on iOS devices (crash events for native code errors).
- Interactions of any application-to-server activity (anything that is supported by the MobileFirst client/server protocol, including push notification).
- Server-side logs that are captured in traditional MobileFirst log files.

MobileFirst Server and MobileFirst Operational Analytics environments can also be built as Docker images that you can deploy and run on the cloud using IBM Containers for Bluemix.

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.

MobileFirst Platform Command Line Interface for iOS

To help you get a better tools experience, IBM MobileFirst Platform Foundation for iOS provides a command-line interface (CLI) tool to easily create and manage apps. By working with the CLI, you can use your preferred text editors or alternative IDEs to create mobile applications.

The commands support tasks such as creating, adding, and configuring with the API library, adding the client-side properties file, and building and deploying the application. From the command line, you can create and deploy adapters, and test them locally. You can administer your project from CLI or REST services, or the MobileFirst Operations Console, where you can control the local server and observe the logs.

IBM API Connect

IBM API Connect is a management solution that addresses all aspects of the application programming interface (API) lifecycle for both on-premises and cloud environments. It offers capabilities to create, run, manage, secure, and monetize APIs and microservices.

With IBM MobileFirst Platform Foundation for iOS V7.1.0 and IBM API Connect V5.0.0, you can complement your mobile strategy with multi-channel API creation and management:

- Create and run APIs for multi-channel systems of engagements, including mobile.
- Add lifecycle management and a layer of governance to your multi-channel APIs.
- Enforce API and mobile security together at the edge of the network with IBM WebSphere® DataPower® Gateways.
- Extend business analytics from points of interaction and mobile server down to the data layer with IBM API Analytics.
- Add developer self-service API onboarding and discovery by using IBM API Connect Developer Portal.

For more information, see the IBM API Connect V5.0.0 product documentation.

Product editions

IBM MobileFirst Platform Foundation for iOS is available in one edition.

IBM MobileFirst Platform Foundation Consumer and IBM MobileFirst Platform Foundation Enterprise

IBM MobileFirst Platform Foundation Consumer and IBM MobileFirst Platform Foundation Enterprise are supported through an IBM International License Agreement and are available from IBM Passport Advantage®. These two editions differ in licensing model only . IBM MobileFirst Platform Foundation Consumer is licensed by 10,000 addressable devices, and must not be used for employees. IBM MobileFirst Platform Foundation Enterprise is licensed by 200 addressable devices and can be used for any type of user. IBM MobileFirst Platform Foundation Consumer and IBM MobileFirst Platform Foundation Enterprise contain the following components:

- An IBM-supported version of the IBM MobileFirst Platform Foundation Developer Edition
- A separate IBM MobileFirst Platform Server component, which is available as an IBM Installation Manager package

This edition contains the following components:

- IBM MobileFirst Platform Command Line Interface for iOS, which is available as an installable download.
- IBM MobileFirst Platform Server component, which is available as an IBM Installation Manager package.

System requirements

System requirements for IBM MobileFirst Platform Foundation for iOS include operating systems, SDKs, and other software.

IBM MobileFirst Platform Foundation for iOS has a number of system requirements that must be met for you to install and configure the product successfully. The system requirements include the following items:

- Operating systems that support IBM MobileFirst Platform Foundation for iOS, including mobile device operating systems
- Required hardware configuration
- Supported software development kits (SDKs)
- Application servers, database management systems, and other software that are required or supported by IBM MobileFirst Platform Foundation for iOS

System requirements by type (high-level)

The requirements in the following links are organized by high-level categories:

- Operating systems
- Software
- Hardware

System requirements by platform (detail)

The requirements in the following links are organized by installation target platform:

- AIX®
- Linux
- Mac OS
- Mobile OS
- Solaris
- Windows

System requirements by component (detail)

The requirements in the following links are organized by product component:

- IBM MobileFirst Platform Application Center
- IBM MobileFirst Platform Cloudant Data Layer Local Edition
- IBM MobileFirst Platform for IBM Containers
- IBM MobileFirst Platform Foundation System Patterns

- IBM MobileFirst Platform Operational Analytics
- IBM MobileFirst Platform Server
- IBM MobileFirst Platform Command Line Interface for iOS
- IBM MobileFirst Platform Studio
- IBM MobileFirst Platform Test Workbench
- IBM MobileFirst Platform Device Runtime

Licensing in MobileFirst Server

The IBM MobileFirst Platform Server supports two different licensing methods based on what you have purchased.

If you have purchased *Perpetual licenses*, you can consume what you have purchased and verify your usage and compliance through the **License tracking page** in the MobileFirst Operations Console and through “License Tracking report” on page 13-130. If you have purchased *Token licenses*, configure your MobileFirst Server to communicate with a remote token license server.

Token Licensing

In a token environment, every product consumes a predefined token value per license, compared to a traditional floating environment where a predefined quantity per license is consumed. The license key has a pool of tokens from which the license server calculates the tokens that are checked in and checked out. Tokens are either consumed or released when a product checks in or checks out licenses from the license server.

Your licensing contract defines whether you might be able to use token licensing, the number of tokens available, and features that are validated by tokens. See “Token license validation” on page 13-133.

If you have purchased token-based licenses, install a version of the MobileFirst Server that supports token licenses and configure your application server so that your server can communicate with the remote token server. See “Installing and configuring for token licensing” on page 6-115.

With token licensing, you can specify the license app type in the application descriptor of each one of your apps before deploying them. The license app type can be either `APPLICATION` or `ADDITIONAL_BRAND_DEPLOYMENT`. For testing, you can set the value of the license app type to `NON_PRODUCTION`. For more information, see “Token licensing: setting the license app type” on page 8-21.

The Rational® License Key Server Administration and Reporting tool that is released with Rational License Key Server 8.1.4.9 can administer and generate reports for the license consumed by IBM MobileFirst Platform Foundation for iOS. You can identify the relevant parts of the report by the following display names: **Mobile First Platform Foundation Application** or **Mobile First Platform Additional Brand Deployment**. These names refer to the license app type for which the tokens are consumed. For more information, see Rational License Key Server Administration and Reporting Tool overview and Rational License Key Server Fix Pack 9 (8.1.4.9).

For information on planning to use token licensing with MobileFirst Server, see “Planning for the use of token licensing” on page 6-23.

To obtain the license keys for IBM MobileFirst Platform Foundation for iOS, you need to access IBM Rational License Key Center. For more information about generating and managing your license keys, see IBM Support - Licensing.

Matrix of features and platforms

IBM MobileFirst Platform Foundation for iOS provides many features and supports many platforms.

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

Consider compatibility with earlier releases before you decide what version of each product component to install. For more information, see “Version compatibility” on page 7-1.

Accessibility

Accessibility features enable people with disabilities, such as restricted mobility and limited vision, to work successfully with IBM MobileFirst Platform Foundation for iOS.

This product supports the following accessibility features:

Magnification

You can use the settings of display systems to magnify development interfaces and supporting documentation.

Screen reading

You can use a screen reader with a digital speech synthesizer, such as IBM Easy Web Browsing, to hear what is displayed on your screen. Consult the documentation with your assistive technology for details on using it with this product and its documentation.

Technical documentation

IBM knowledge center is a browser-based environment that you can access by using a web browser. The HTML files that are supported by browsers are validated for accessibility. The equivalent PDF file for the same content is provided for convenience only and is not fully accessible.

Accessibility of web app consoles

To navigate web application consoles, use the accessibility features of your browser.

You can navigate through the different MobileFirst environments and their documentation by using keyboard shortcuts. Eclipse provides accessibility features for its development environments. Internet browsers also provide accessibility features for web applications, such as the IBM MobileFirst Platform Operations Console, the IBM MobileFirst Platform Application Center console, and the IBM MobileFirst Platform Application Center mobile client.

If you are using the latest interim fix of IBM MobileFirst Platform Foundation for iOS, the IBM MobileFirst Platform Operational Analytics Console also provides accessibility features.

Accessibility of the MobileFirst Platform Command Line Interface for iOS

For accessibility, you can turn off the multicolor display of output that is provided by the MobileFirst Platform Command Line Interface for iOS.

By default, status messages that are displayed by the MobileFirst Platform Command Line Interface for iOS use various colors to indicate success, errors, and warnings. You can use the **`--no-color`** option on any MobileFirst Platform Command Line Interface for iOS command to suppress the use of these colors for that command. When **`--no-color`** is specified, output is displayed in the text display colors that are set for your operating system console.

Accessibility of IBM MobileFirst Platform Foundation for iOS installation and configuration

The installation and configuration of IBM MobileFirst Platform Foundation for iOS can be carried out by using command-line and Ant tasks.

There are two ways to install and configure IBM MobileFirst Platform Foundation for iOS: by graphical user interface (GUI), or by command-line.

Although the graphical user interface (IBM Installation Manager in wizard mode or Server Configuration Tool) does not provide information about user interface objects, equivalent function is available with the command-line interface. All the functions in the GUI are supported through the command-line, and some particular installation and configuration features are only available with the command-line.

The following topics provide you with the information on how the installation and configuration can be done without GUI:

- "Working with sample response files for IBM Installation Manager" on page 6-35

  This method enables silent installation and configuration of MobileFirst Server and Application Center. You have the possibility to not install Application Center by using the response file named `install-no-appcenter.xml`. You can then use Ant task to install it at a later stage. See "Installing the Application Center with Ant tasks" on page 6-216. In this case, the installation and the upgrading of Application Center can be done independently.
- "Using Ant tasks to install MobileFirst Server administration" on page 6-62
- "Using Ant tasks to deploy the project WAR file" on page 11-15
- "Installing the Application Center with Ant tasks" on page 6-216
- "Installing MobileFirst Operational Analytics with Ant tasks" on page 6-191

  Note: MobileFirst Operational Analytics can be installed only by Ant tasks, there is no GUI equivalent.

Release notes

You can identify the latest information about this product release and all its fix packs.

What's new in V7.1.0

Discover the new features and changes in IBM MobileFirst Platform Foundation for iOS V7.1.0 compared to the previous version of this product.

Portability between IBM Bluemix and on-premises IBM MobileFirst Platform Foundation for iOS

New features help you to move your app between the cloud and your on-premises platforms. This portability gives you the flexibility to develop and test your app in one environment, and deploy your app to production in another.

IBM MobileFirst Platform Foundation for iOS on IBM Containers

You can run MobileFirst applications in the cloud using IBM Containers on Bluemix.

The evaluation of MobileFirst Platform Foundation on IBM Containers (available from IBM developerWorks®) is for MobileFirst users who want to work with apps and images locally and deploy images to the IBM Containers service on Bluemix. Using this evaluation offering, you can quickly deploy and run mobile apps without the need for IT involvement but some set up and configuration is required. To use the evaluation, see Run Foundation on Bluemix tutorial.

Note: For production-level use, see "IBM MobileFirst Platform Foundation for iOS on IBM Containers" on page 3-11 in the list of V7.1.0 interim fixes.

Migrate IBM Bluemix apps to IBM MobileFirst Platform Foundation for iOS

You can now migrate apps that are developed on IBM MobileFirst Platform for iOS on IBM Bluemix to IBM MobileFirst Platform Foundation for iOS in a few steps. A new API translates the IBM MobileFirst Platform for iOS on Bluemix API to the IBM MobileFirst Platform Foundation for iOS iOS API. For more information, see Migrating iOS apps from Bluemix. For more information about the API, see "Objective-C client-side API for migrated Bluemix iOS apps" on page 10-2.

REST API compatibility for server-side code

Back-end server applications deployed outside of MobileFirst Server can now interact directly with some Push service functions through new REST APIs. The new Push REST API for the MobileFirst runtime makes it more convenient for server applications to integrate with the Push capabilities of IBM MobileFirst Platform Foundation for iOS because you no longer need to develop server-side adapter code to expose those functions.

Enhanced enterprise resilience

Added capabilities let you better handle dynamic workloads and deal with disaster recovery.

Running application in session-independent mode

Session-independent mode was introduced as the *default* mode for MobileFirst Server, starting with V7.1.0.

The session-independent mode decouples the link between authentication context and HTTP sessions by using an attribute store module for caching the authentication context. In this mode, work between the client and MobileFirst Server is session-independent, allowing IT to auto-scale the network as needed and use a load-balancer without requiring sticky HTTP sessions.

For more information, see "Session-independent mode" on page 8-108.

Important: By default, new projects that are created in V7.1.0 are *session independent*. Old projects, even after upgrade, remain session-dependent. For more information about upgrading older apps to work in session-independent mode in V7.1.0, see Upgrading projects to work in session-independent mode.

Disaster recovery: active-active topology

If your system is operating in session-independent mode, you can provide high availability and disaster recovery to your distributed network. The topology involves several instances of IBM MobileFirst Platform Server that are deployed in two or more active data centers, in combinations with IBM Cloudant and IBM WebSphere eXtreme scale. Optionally, this configuration might use a load-balancer to determine which site handles the client requests.

For more information, see "Disaster recovery: active-active topology" on page 6-136.

Using Cloudant as a supporting database for MobileFirst Server and MobileFirst Server administration

When you install IBM MobileFirst Platform Foundation, you are now able to use Cloudant as your database management system for MobileFirst Server and MobileFirst Server administration. You can now benefit from a NoSQL local or Cloud-based Cloudant database for your MobileFirst runtime and administration.

To use a Cloudant database, you must have an account at `cloudant.com` that uses the Cloudant database in the Cloud, or you must install IBM MobileFirst Platform Cloudant Data Layer Local Edition, as described in the Cloudant Data Layer Local Edition user documentation.

For more information about how to use Cloudant as your database, see:

- "Planning the creation of the databases" on page 6-19
- "Creating the Cloudant database for MobileFirst Server administration" on page 6-51
- "Creating the Cloudant database" on page 11-9

Improved development experience

New APIs, frameworks, capabilities, and tools make developing apps with IBM MobileFirst Platform Foundation for iOS simpler and help make developers more productive.

Streamlined experience for developing native apps

As a native app developer, you can take advantage of streamlined methods to get the IBM MobileFirst Platform Foundation SDK, as well a simplified workflow in your native app development environment.

Use `mfp push` to accelerate development

With V7.1, the IBM MobileFirst Platform Command Line Interface for iOS (CLI) provides a new command, **`mfp push`**, that enables you to register, build and deploy your app all with a single command. With **`push`**, you can deploy to remote as well as local instances of the MobileFirst Server. You can now proceed through the entire MobileFirst development process by using only the CLI and your preferred source code editor or IDE. For more information, see "**`push`**" on page 15-9.

Enhanced `mfp config` simplifies app configuration

The V7.1 IBM MobileFirst Platform Command Line Interface for iOS (CLI) also provides an enhanced **`mfp config`** command. With these enhancements, you can configure MobileFirst features without requiring manual editing of the application descriptor file. For more information, see "**`config`**" on page 15-4.

Use CocoaPods to get the IBM MobileFirst Platform Foundation iOS SDK

You can now use CocoaPods to get the IBM MobileFirst Platform Foundation iOS SDK. For more information, see "Adding the IBM MobileFirst Platform Foundation iOS SDK to a new application with CocoaPods" on page 8-13.

iOS libraries now packaged as frameworks

Starting with IBM MobileFirst Platform Foundation for iOS 7.1.0, the iOS Client SDK static libraries are packaged as iOS frameworks.

To develop native iOS apps that were originally developed in IBM MobileFirst Platform Foundation for iOS, version 7.0.0 or earlier in IBM MobileFirst Platform Foundation for iOS, version 7.1.0, you must modify your code. For more information, see "Upgrading existing native iOS applications to IBM MobileFirst Platform Foundation for iOS version V7.1.0 manually" on page 7-6.

Additional services and other development enhancements

Various new features and services provide additional functions and capabilities for you to build, test, deploy, and monitor enterprise-grade apps.

Discovery of OData and Azure services

Starting with V7.1, the discovery of both basic OData services and Microsoft Azure OData services are supported. You can use adapters to access these back-end services. For more information, see "Microsoft Azure OData adapters" on page 8-83 and "OData adapters" on page 8-83.

Liberty Server upgrade

Starting with V7.1, WebSphere Application Server Liberty Core that is embedded with MobileFirst Studio and the MobileFirst Platform Command Line Interface for iOS has been upgraded from V8.5.5.4 to V8.5.5.5.

Improved data reach with Cloudant and JSONStore

Leverage Cloudant and JSONStore to keep mobile app responsive when they are off the network, while taking advantage of on-device encryption to keep data secure.

Encypt device data with Cloudant

You can encrypt data on Android or iOS devices with the SQLCipher open source extension to the SQLite library. This library provides transparent 256-bit Advanced Encryption Standard (AES) encryption of database files. To use SQLCipher commercially, you must obtain the necessary license.

Cloudant does not support Federal Information Processing Standard (FIPS) 140-2 compliance at this time.

For more information, see “Encrypting data on the device” on page 8-155.

64-bit support for JSONStore on iOS

JSONStore is supported on 64-bit architecture on iOS. For more information, see “JSONStore” on page 8-125.

Hardened security features

New IBM MobileFirst Platform Foundation for iOS security features strengthen the security of your app and its communication with back-end systems.

Certificate pinning

Version 7.1 supports the use of certificate pinning, which can help to reduce man-in-the-middle attacks on communications that are secured with the SSL/TLS protocol. With certificate pinning, your mobile client accepts only a specific digital certificate, instead of any certificate that corresponds to a trusted CA root certificate. For more information, see “Certificate pinning” on page 8-229.

64-bit support for FIPS 140-2 on iOS

Starting with Version 7.1, FIPS 140-2 is supported on 64-bit architecture on iOS. For more information, see “FIPS 140-2 support” on page 12-125.

Secure access of corporate back ends to protected resources

Starting with Version 7.1, organizations can let a confidential (non-mobile) client connect to mobile services in a secure way.

The confidential client flow is based on the client credentials flow, as defined in the OAuth specification. The client authenticates with the authorization server using a certificate. The authentication server validates the certificate, and if it is valid, issues an access token to the client. The certificate has to be signed by a trusted signer certificate that is configured at the authorization server.

For more information, see “Exposing mobile services to non-mobile (confidential) clients” on page 8-209.

Easier deployment and management of IBM MobileFirst Platform Foundation for iOS and your apps

Various new tools and other improvements make it easier for you to deploy and manager IBM MobileFirst Platform Foundation for iOS and your apps.

Improvements to the MobileFirst Analytics Console

In V7.1.0, the user interface of the MobileFirst Analytics Console was improved to provide an enhanced user experience for viewing your analytics data. For more information, see “Navigating the Analytics Console” on page 13-17.

The improved MobileFirst Analytics Console also requires migration of your analytics data. For more information, see “Migration to the V7.1.0 Analytics Console” on page 7-12.

Improvements to the MobileFirst Operations Console

Changes to the interface controls and the behavior of MobileFirst Operations Console are intended to provide an enhanced user experience for deploying and managing your applications. See for example:

- “Deploying applications and adapters to MobileFirst Server” on page 11-84
- “Deleting apps” on page 11-86
- “Defining administrator messages from MobileFirst Operations Console in multiple languages” on page 12-5

Easier upgrade process with the Server Configuration Tool

You can now use the Server Configuration Tool to upgrade a configuration of MobileFirst Server that was previously deployed with the Server Configuration Tool, from IBM Worklight® Foundation V6.2.0, or later.

For more information, see “Upgrading with the Server Configuration Tool” on page 7-72.

Asymmetric deployment in a WebSphere Application Server Network Deployment cell

You can deploy MobileFirst administration components and runtimes asymmetrically in different clusters of a WebSphere Application Server Network Deployment cell. Therefore, the administration components are deployed in a different Java Virtual Machine (JVM) from the runtimes.

For more information, see “WebSphere Application Server Network Deployment topologies” on page 6-15.

Asymmetric deployment of MobileFirst administration components and runtimes is only supported with WebSphere Application Server Network Deployment.

Removal of the reports database from the installation process

The reports database is a deprecated feature, and is no longer part of the installation of the product by default. You should use IBM MobileFirst Platform Operational Analytics instead.

However, if you still need it, you can perform extra steps to install, configure, and upgrade the reports database. For more information, see “Manually creating and configuring the reports database” on page 13-117, and “Upgrading the reports database” on page 13-121.

Installation of IBM MobileFirst Platform Operational Analytics with Ant tasks

You can now use Ant tasks to install MobileFirst Operational Analytics and deploy it to your application server. For more information, see the following topics:

- “Installing MobileFirst Operational Analytics with Ant tasks” on page 6-191
- “Ant tasks for installation of MobileFirst Operational Analytics” on page 15-54
- “Sample configuration files for MobileFirst Operational Analytics” on page 15-71

If you installed MobileFirst Operational Analytics with Ant tasks, you can also use Ant tasks to upgrade this component to a fix pack, or interim fix. For more information, see “Applying a fix pack to IBM MobileFirst Platform Operational Analytics” on page 7-85.

Support of the Addressable Device metric

In IBM MobileFirst Platform Foundation V7.1.0, from the MobileFirst Operations Console, you can see the number of addressable devices in the current calendar month for applications that you can define for IBM MobileFirst Platform Foundation Consumer, or for IBM MobileFirst Platform Foundation Enterprise. The License Tracking report tracks devices that run native or hybrid applications on iOS, Android, or Windows.

For more information, see “License tracking” on page 13-128.

For more information about how to set the target category for your application, see the topic about application descriptor for iOS.

Multicultural support in MobileFirst Operations Console

Multicultural support has been added to MobileFirst Operations Console to make it easier to deploy and manage applications in languages other than English. MobileFirst Operations Console is translated into the following languages: Brazilian Portuguese, French, German, Italian, Japanese, Korean, Russian, Simplified Chinese, Spanish, and Traditional Chinese. Dates and numbers are now formatted according to the appropriate cultural environment.

Installation of the Application Center with Ant tasks

In V7.1.0, you can use Ant tasks to install and upgrade Application Center.

For more information, see:

- “Installing the Application Center with Ant tasks” on page 6-216
- “Ant tasks for installation of Application Center” on page 15-45

- “Applying a fix pack to Application Center installed with Ant tasks” on page 7-86

Improved MobileFirst API

New features improve and extend the APIs that you can use to develop mobile applications.

New client-side API for migrated Bluemix iOS apps

The Objective-C client-side API for migrated apps from IBM Bluemix provides a compatibility layer that translates the code that is in your original Bluemix app to code that is recognized by IBM MobileFirst Platform Foundation for iOS. For more information, see “Objective-C client-side API for migrated Bluemix iOS apps” on page 10-2.

Updated JavaScript server-side API

IBM MobileFirst Platform Foundation for iOS V7.1.0 includes updates in its JavaScript server-side API that you can use to extend the MobileFirst Server, as follows:

New API:

getClientId()

With this new API, you can return the String value of the ClientID associated with this client.

For more information, see the WL.Server class.

For more information, see “JavaScript server-side API” on page 10-4.

Updated Objective-C client-side API for iOS

IBM MobileFirst Platform Foundation for iOS V7.1.0 includes updates in its Objective-C client-side API to develop native apps on iOS, as follows:

Updated API classes:

WLClient

- The following functions have been enhanced:
  - logout:withDelegate:options
  - logout:withDelegate

  The logout mechanism in this version has changed. In addition to calling the logout in the server, both logout functions delete the access token and ID token from the client device. The next time the OAuth flow starts, the client is required to again log in to the realm from which it was logged out.
- The pinTrustedCertificatePublicKeyFromFile method has been added to support certificate pinning.

For more information, see WLClient class

Updated Java server-side API

IBM MobileFirst Platform Foundation for iOS V7.1.0 includes updates in its Java server-side API that you can use to extend the MobileFirst Server, as follows:

New API:

getClientId()

With this new API, you can return a unique identifier associated with this client.

For more information, see the OAuthSecurityContext interface.

For more information, see “Java server-side API” on page 10-4.

What's new in V7.1.0 interim fixes

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

Interim fixes are cumulative. When you download the latest V7.1.0 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.

Where an APAR number is listed, you can confirm that an interim fix has that feature by searching the interim fix README file for that APAR number.

Token licensing (APAR PI48649)

Token licensing was introduced with interim fix 7.1.0.0-MFPF-IF201509132345 and is also included when you download IBM MobileFirst Platform Foundation for iOS V7.1.0 from Passport Advantage after 15 September 2015.

Token licensing overview

Interim fixes 7.1.0.0-MFPF-IF201509132345 and later provide IBM MobileFirst Platform Foundation for iOS with an alternative licensing method that is based on tokens.

In a token environment, every product consumes a predefined token value per license. Licensing is handled by a license server that has a pool of tokens from which the license server calculates the tokens that are checked in and checked out. Tokens are either consumed or released when a product checks in or checks out licenses from the license server (Rational License Key Server).

Your product contract defines whether you must use token licensing or not. The contract defines the number of tokens available and the features that are validated by tokens. If you purchased token licenses, you must install a version of MobileFirst Server that supports token licensing and also configure your application server so that it communicates with the remote license server.

For each application, you must specify the license app type in the application descriptor before deployment. The license app type can be `NON_PRODUCTION`, `APPLICATION`, or `ADDITIONAL_BRAND_DEPLOYMENT`.

For more information, see the following topics:
- "Licensing in MobileFirst Server" on page 2-10
- "Installing and configuring for token licensing" on page 6-115
- "Token licensing: setting the license app type" on page 8-21

Important:
- You cannot activate token licensing in IBM MobileFirst Platform Foundation for iOS V7.1.0 and any interim fixes that are delivered before 14 September 2015. If the graphic mode installation does not display the **Token Licensing** panel, or if the silent installation with `imcl` does not confirm that token licensing is activated, then token licensing is not activated. The installation is then not compliant with token licensing. You must download IBM MobileFirst Platform Foundation for iOS V7.1.0 from Passport Advantage after 15 September 2015, or download and apply interim fix 7.1.0.0-MFPF-IF201509132345 or later.
- Rational License Key Server 8.1.4.8 or later must be installed and configured. For more information, see Rational License Key Server Portal.
- Make sure that the license keys for IBM MobileFirst Platform Foundation for iOS are generated. For more information about generating and managing your license keys with IBM Rational License Key Center, see IBM Support - Licensing.

iOS 9 support

If you use Xcode 7 to compile your apps, or if you use extended authenticity protection for your apps, install the latest interim fix and review the following sections to ensure that your apps continue to work on iOS 9.

If you build your apps with Xcode 6.4 and do not use extended authenticity protection, you can rebuild your apps with iOS 9 support without installing the latest IBM MobileFirst Platform Foundation for iOS interim fix.

Extended authenticity checking for apps that undergo app thinning

App thinning was introduced by Apple in iOS 9 for apps that are compiled with Xcode 7. App thinning reduces the size of files that are downloaded from the App Store. The feature might affect the extended authenticity features of IBM MobileFirst Platform Foundation for iOS apps because the binary file in the App Store might differ from the one that is downloaded to the client device.

After you apply latest interim fix, the app thinning feature is available for any app that uses iOS 9 and later and Xcode 7 and later and that was created by using IBM MobileFirst Platform Foundation for iOS V7.0.0 and later.

For more information, see "Enabling extended authenticity checking for apps that undergo app thinning" on page 11-57.

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 MobileFirst Platform Foundation for iOS. To use the MobileFirst 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 MobileFirst Platform Foundation for iOS.

For more information, see "Disabling bitcode in Xcode builds" on page 8-20.

Support for dynamic `.tbd` libraries in Xcode 7

Xcode 7 replaces dynamic `.dylib` libraries with more lightweight `.tbd` files. Up to now, IBM MobileFirst Platform Foundation for iOS 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:

- "Upgrading existing native iOS applications to IBM MobileFirst Platform Foundation for iOS version V7.1.0 manually" on page 7-6
- "Copying SDK and configuration files from the project" on page 8-11
- Modifying code in Bluemix apps

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 MobileFirst Platform Foundation for iOS V6.0.0 and later automatically turn off transport security to allow all non-secure connections to the MobileFirst Development Server.

For more information, see "Enforcing TLS-secure connections in iOS apps" on page 8-19.

New features in the MobileFirst Analytics Console

If you are using the latest interim fix of IBM MobileFirst Platform Foundation for iOS, you can use the new features in the MobileFirst Analytics Console.

Alert definitions with thresholds (APAR PI50260)

You can set thresholds in alert definitions in the MobileFirst Analytics Console to better monitor your activities.

For more information, see "Alerts" on page 13-23.

Exporting and importing custom chart definitions (APAR PI50263)

You can export and import your custom chart definitions programmatically or in the MobileFirst Analytics Console.

For more information, see "Exporting and importing custom chart definitions" on page 13-66.

Application crash analytics (APAR PI53648)

You can quickly see information about your application crashes in the Dashboard page of the MobileFirst Analytics Console. Two new charts are available: **Crash Overview** and **Crashes**.

For more information, see "Application crash monitoring" on page 13-36.

To better administer your apps, you can view app crash analytics. In the Dashboard page of the MobileFirst Analytics Console, a **Crash Summary** tab is available.

For more information, see "Application crash troubleshooting" on page 13-37.

You can create an alert definition to receive an alert when the number of app crashes or the crash rate for your applications exceeds your defined threshold.

For more information, see "Creating an alert definition for application crashes" on page 13-32.

IBM MobileFirst Platform Foundation for iOS on IBM Containers

In addition to the evaluation version of IBM MobileFirst Platform Foundation for iOS on IBM Containers released for V7.1 (available from IBM developerWorks), IBM MobileFirst Platform Foundation for iOS on IBM Containers is now available for production use with the interim fix update available on the IBM Passport Advantage site.

The interim fix for APAR PI55391 on Fix Central, applied to IBM MobileFirst Platform Foundation for iOS on IBM Containers V7.1 (available from IBM Passport Advantage site) supports IBM MobileFirst Platform Foundation for iOS to be deployed on container groups.

For details on applying the interim fix refer to Applying IBM MobileFirst Platform Foundation for iOS interim fixes in an IBM Containers environment.

Note: The updated *Prerequisites* section in IBM MobileFirst Platform Foundation for iOS on IBM Containers is applicable for IBM MobileFirst Platform Foundation for iOS on IBM Containers with the interim fix for APAR PI55391.

Learn more about "Deploying to the cloud in an IBM Container" on page 11-107.

JSONStore framework for MobileFirst iOS applications (APAR PI50359)

After you install the latest interim fix of IBM MobileFirst Platform Foundation for iOS, JSONStore becomes an optional feature for native MobileFirst iOS applications. To upgrade your hybrid iOS applications from a previous version, you must complete some additional steps.

JSONStore framework for native MobileFirst iOS applications

To use JSONStore in a native MobileFirst iOS application, you must import the `<IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.h>` umbrella header file. You can obtain the JSONStore feature in two ways:

- You can import the JSONStore framework from the worklight api folder. For more information, see “Copying SDK and configuration files from the project” on page 8-11.
- You can use CocoaPods. For more information, see “Enabling JSONStore” on page 8-129.

For more information about creating native MobileFirst iOS applications, see “Developing native applications” on page 8-7.

CloudantToolkit and IMFData frameworks are deprecated

Use the CDTDatastore SDK as a replacement for CloudantToolkit and IMFData frameworks.

If you want to access remote stores directly, use REST calls in your application and refer to the Cloudant API Reference.

For more information about how to use Cloudant Sync in your mobile apps, see “Storing mobile data in Cloudant” on page 8-151.

Accessibility enhancements

Accessibility improvements were made to the IBM MobileFirst Platform Command Line Interface for iOS and to the technical documentation.

IBM MobileFirst Platform Command Line Interface for iOS

To improve accessibility, a **`--no-color`** option that suppresses the use of varying text colors was added to the IBM MobileFirst Platform Command Line Interface for iOS. For more information, see “CLI commands usage” on page 8-2.

To install and configure the product without graphical user interface, see “Accessibility of IBM MobileFirst Platform Foundation for iOS installation and configuration” on page 2-12. For more information about the accessibility features, see “Accessibility” on page 2-11.

Technical documentation

The IBM MobileFirst Platform Foundation for iOS documentation is accessible when you are using screen reading software. The IBM Knowledge Center documentation is hosted in the IBM Knowledge Center - Hosted Edition service. To request the current accessibility status for the IBM Knowledge Center - Hosted

Edition service, see IBM Product Accessibility information and select the entry for IBM Knowledge Center - Hosted Edition.

Enhanced instructions for upgrading MobileFirst Server to V7.1.0

A series of steps-by-steps topics is introduced to guide you through the entire upgrading process of your MobileFirst Server in production. Each topic provides you a complete procedure to upgrade from a specific version to V7.1.0:

- Applying a fix pack for MobileFirst Server V7.1.0
- Upgrading MobileFirst Server from V7.0.0 to V7.1.0
- Upgrading MobileFirst Server from V6.3.0 to V7.1.0
- Upgrading MobileFirst Server from V6.2.0 to V7.1.0
- Upgrading MobileFirst Server from V6.1.0 to V7.1.0
- Upgrading MobileFirst Server from V6.0.0 to V7.1.0
- Upgrading MobileFirst Server from V5.0.6 to V7.1.0

For more information, see “Upgrading to MobileFirst Server V7.1.0 in a production environment” on page 7-13.

LDAP login module changes

The LDAPLoginModule class was modified to add a `searchUniqueUser` **`validationType`** configuration-parameter value, and the following related configuration parameters were added: **`ldapSearchUserName`**, **`ldapSearchPassword`**, **`ldapUseAttributeToExtract`**, and **`ldapAttributeToExtract`**. For more information, see searchUniqueUser.

IBM WebSphere DataPower integration enhancements

The support for integrating with IBM WebSphere DataPower as a reverse proxy and security gateway was improved. A preconfigured DataPower pattern, related MobileFirst samples, and detailed documentation for integrating with DataPower by using LTPA and DataPower HTML forms-based authentication are provided. For more information, see “Integrating with DataPower as a reverse proxy using LTPA and form-based authentication” on page 6-168.

IBM Trusteer integration enhancements (APAR PI67861)

It is now possible to disable the automatic initialization of IBM Trusteer Mobile SDK instances that are integrated with MobileFirst projects. When the automatic initialization is disabled, you can initialize the Trusteer Mobile SDK manually by using either the Trusteer Mobile SDK or the MobileFirst client-side API. For more information, see “Integrating IBM Trusteer for iOS” on page 14-10.

IBM MobileFirst Platform Foundation for iOS on IBM Containers

IBM MobileFirst Platform Foundation for iOS on IBM Containers now supports enterprise dashDB database on IBM Bluemix.

Important: Enterprise Transactional Plans that support OLTP workloads are the only supported dashDB plans.

Note: See prerequisites for deploying IBM MobileFirst Platform Foundation for iOS on IBM Containers.

Learn more about Deploying to the cloud in an IBM Container.

Deprecated and removed features

If you are migrating from an earlier release of the product, be aware of the various features that have been deprecated or removed in this and earlier releases.

Deprecated features

Definition: Pertaining to an entity, such as a programming element or feature, that is supported but no longer recommended and that might become obsolete.

For a list of deprecated features, see “Deprecated features and API elements”

Removed features

Definition: Pertaining to a feature that is no longer included in a product.

For a list of removed features, see “Removed features” on page 3-15

Deprecated features and API elements

The following API elements are deprecated from this release.

Features deprecated in V7.1.0

Table 3-1. Features deprecated in V7.1.0.

Category	Deprecation	Recommended Action
iOS 6	The iOS 6 environment is deprecated.	
mfp bd	This CLI command is deprecated.	Use **mfp push** instead.
mfp build	This CLI command is deprecated.	Use **mfp push** instead.
mfp create-server	This CLI command is deprecated.	Use **mfp server create** instead.
mfp deploy	This CLI command is deprecated.	Use **mfp push** instead.

Features deprecated in V7.0.0

Table 3-2. Features deprecated in V7.0.0.

Category	Deprecation	Recommended Action
Analytics Reports database	The Reports database, often referenced as `WLREPORT` in the documentation, is deprecated in IBM MobileFirst Platform Foundation for iOS V7.0.0.	Use IBM MobileFirst Platform Operational Analytics instead. Note that setting up the Reports database is optional in this release and earlier releases. Also note that the use of the Reports database is redundant with MobileFirst Operational Analytics in this release and recent earlier releases.

Table 3-2. Features deprecated in V7.0.0 (continued).

Category	Deprecation	Recommended Action
Analytics BIRT predefined reports	The predefined BIRT reports are deprecated.	Use IBM MobileFirst Platform Operational Analytics console and custom chart support instead.
The JAR files and JavaScript libraries that enable SSO between IBM MobileFirst Platform Foundation for iOS and other external services	The `external-server-libraries` directory and its contents are deprecated. The following API URLs are also deprecated: `<application root context>/oauth/*`	Use the MobileFirst OAuth-based security model instead. For more information about this model, see "OAuth-based security model" on page 8-188.

API elements deprecated in V7.0.0

Table 3-3. API elements deprecated in V7.0.0.

Category	Deprecation	Recommended Action
WLClient	[WLCLient lastAccessToken]	Use [WLAuthorizationManager cachedAuthorizationHeader] instead.
	[WLCLient lastAccessTokenForScope]	Use [WLAuthorizationManager cachedAuthorizationHeader] instead.
	[WLCLient obtainAccessTokenForScope]	Use [WLAuthorizationManager obtainAuthorizationHeaderForScope] instead.
WLResponse	[WLResponse getResponseJson]	Use the responseJson property instead.

Features deprecated in V6.3.0

Table 3-4. Features deprecated in V6.3.0.

Category	Deprecation	Recommended action
Sencha Touch tools	Sencha Touch tools are deprecated in MobileFirst Studio. Specific Sencha Touch tools might be removed from MobileFirst Studio in a future release. **Note:** Sencha Touch is still supported by the product.	You can continue to use Sencha Touch by copying the relevant files into your MobileFirst application to a directory of your choosing. Then, add the necessary reference tags into the main `index.html` file.

Removed features

The following features are removed from this and earlier releases of this product.

Features removed in V7.1.0

Table 3-5. Features removed in V7.1.0

Feature
The IBM MobileFirst Starter on Bluemix (**ibm-mobilefirst-starter**) which provides a fully functional instance of IBM MobileFirst Platform Foundation for iOS, is now removed from V7.1.0.

Features removed in V7.0.0

Table 3-6. Features removed in V7.0.0.

Feature
The JAR files and JavaScript libraries that enable SSO between IBM MobileFirst Platform Foundation for iOS and other external services are removed. Use the MobileFirst OAuth-based security model instead. For more information about the OAuth-based security model, see "OAuth-based security model" on page 8-188.
The "shake to refresh" feature is removed in IBM MobileFirst Platform Foundation for iOS V7.0.0.

Features removed in V6.3.0

Table 3-7. Features removed in V6.3.0.

Feature
The IBM Worklight Application Framework (beta) set of tools is removed in IBM MobileFirst Platform Foundation for iOS V6.3.0.

Known issues

You can identify the latest known issues and their resolutions, for this product release and all its fix packs, by browsing this dynamic list of documents.

Click the following link to receive a dynamically generated list of documents for this specific release and all its fix packs, including known issues and their resolutions, and relevant downloads: http://www.ibm.com/support/search.wss?tc=SSHSCD&atrn=SWVersion&atrv=7.1

The following websites provide helpful community resources:

- Developer Center for IBM MobileFirst Platform (Help page), where you can post questions to Stack Overflow website, and get answers, by using the following tags:
  - mobilefirst
  - worklight for past releases
- dW Answers website, where you can post questions and get answers, by using the following tags:
  - mobilefirst
  - worklight for past releases

Known limitations

General limitations apply to IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS 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: For more information about product known limitations or issues, see "Known issues" on page 3-16.

Globalization

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

- Part of the product IBM MobileFirst Platform Foundation for iOS V7.1.0, including its documentation, is translated in the following languages: Simplified Chinese, Traditional Chinese, French, German, Italian, Japanese, Korean, Portuguese (Brazil), Russian, and Spanish. Mainly user-facing text is translated.
- The applications that are generated by IBM MobileFirst Platform Foundation for iOS are not fully bidirectional enabled. Mirroring of the graphic user interface (GUI) elements and the control of the text direction are not provided by default. However, there is no hard dependency from the generated applications on this limitation. It is possible for the developers to achieve full bidi compliance by manual adjustments in the generated code.
- Although translation into Hebrew is provided for IBM MobileFirst Platform Foundation for iOS core functionality, some GUI elements are not mirrored.
- Names of projects, apps, and adapters must be composed only of the following characters:
  - Uppercase and lowercase letters (A-Z and a-z)
  - Digits (0-9)
  - Underscore (_)
- There is no support for Unicode characters outside the Basic Multilingual Plane.

IBM MobileFirst Platform 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.

IBM MobileFirst Platform Operational Analytics

The IBM MobileFirst Platform Operational Analytics has the following limitations:

- In Analytics Console, the format for numbers does not follow the International Components for Unicode (ICU) rules.
- In Analytics Console, the numbers do not use the user's preferred number script.

- In Analytics Console, the format for dates, times, and numbers are displayed according to the language setting of the operating system rather than the locale of Microsoft Internet Explorer.
- When you create a custom filter for a custom chart, the numerical data must be in base 10, Western, or European numerals, such as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
- When you create a custom filter for a custom chart, the query of a custom filter does not use Unicode Normalization Forms C (NFC) for comparison. The query does not consider language sensitivity such as normal matching, accent-insensitive, case-insensitive, and 1-to-2 mapping for search function to run correctly in different languages. The message property uses NFC and considers language sensitivity though.
- If you are using the latest interim fix of IBM MobileFirst Platform Foundation for iOS, the numerical data must be in base 10, Western, or European numerals, such as 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, when you create an alert in the Alert Management page of the Analytics Console.
- The Analytics page of the MobileFirst Operations Console supports the following browsers:
  - Microsoft Internet Explorer version 10 or later
  - Mozilla Firefox ESR or later
  - Apple Safari on iOS version 7.0 or later
  - Google Chrome latest version

IBM MobileFirst Platform Operations Console

The MobileFirst Operations Console has the following limitations:

- Provides only partial support for bidirectional languages.
- The text direction cannot be changed when notification messages are sent to an Android device :
  - if the first letters typed are in a right-to-left language, such as Arabic and Hebrew, the whole text direction is automatically right-to-left.
  - if the first letters typed are in a letf-to-right language, the whole text direction is automatically left-to-right.
- Some error messages of the MobileFirst Operations Console are not translated because they are generated at the server side.

IBM MobileFirst Platform Command Line Interface for iOS

The MobileFirst Platform Command Line Interface for iOS has the following limitations:

- Provides only partial support for bidirectional languages.
- The dates and numbers might not be formatted according to the locale.

Server Configuration Tool

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 Server Configuration Tool has the same globalization limitation as the Windows console to display strings that are out of the default code page.

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.
- 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 must be kept short to avoid truncation at the edge of the screen.
- The applications that are developed with MobileFirst Application Framework running in Portuguese (Portugal) will see runtime messages in Portuguese (Brazil).
- JSONStore does not handle normalization. The Find functions for the JSONStore API do 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.

Adapters

Third-party adapter dependencies

MobileFirst Server comes packaged with the following third-party dependencies. Ensure that you do not package different versions of those dependencies in your project.

- commons-cli:commons-cli:jar:1.2
- commons-fileupload:commons-fileupload:jar:1.3.1
- com.fasterxml.jackson.core:jackson-databind:jar:2.2.2
- com.fasterxml.jackson.core:jackson-annotations:jar:2.2.2
- com.fasterxml.jackson.core:jackson-core:jar:2.2.2
- com.fasterxml.jackson.jaxrs:jackson-jaxrs-json-provider:jar:2.2.2
- com.fasterxml.jackson.jaxrs:jackson-jaxrs-base:jar:2.2.2
- com.fasterxml.jackson.module:jackson-module-jaxb-annotations:jar:2.2.2
- commons-logging:commons-logging:jar:1.1.3
- org.codehaus.jackson:jackson-core-asl:jar:1.9.13
- org.codehaus.jackson:jackson-mapper-asl:jar:1.9.13
- commons-dbcp:commons-dbcp:jar:1.4
- commons-pool:commons-pool:jar:1.6
- commons-codec:commons-codec:jar:1.9
- com.thoughtworks.xstream:xstream:jar:1.2.2.osgi
- xpp3:xpp3_min:jar:1.1.3.4.Oosgi
- org.springframework:spring-beans:jar:3.2.13.RELEASE
- org.springframework:spring-core:jar:3.2.13.RELEASE
- org.springframework:spring-jdbc:jar:3.2.13.RELEASE
- org.springframework:spring-tx:jar:3.2.13.RELEASE
- org.springframework:spring-web:jar:3.2.13.RELEASE

- org.springframework:spring-aop:jar:3.2.13.RELEASE
- aopalliance:aopalliance:jar:1.0.osgi
- org.springframework:spring-orm:jar:3.2.13.RELEASE
- org.springframework:spring-context:jar:3.2.13.RELEASE
- org.springframework:spring-expression:jar:3.2.13.RELEASE
- org.slf4j:slf4j-api:jar:1.6.1
- org.slf4j:slf4j-jdk14:jar:1.6.1
- commons-io:commons-io:jar:2.4
- commons-lang:commons-lang:jar:2.6
- org.apache.commons:commons-lang3:jar:3.3.2
- oro:oro:jar:2.0.8.v201005080400
- org.eclipse.wst.jsdt:debug.rhino.debugger:jar:1.0.400
- org.eclipse.wst.jsdt:debug.core:jar:3.1.100
- org.eclipse.wst.jsdt:debug.transport:jar:1.0.200
- org.apache.httpcomponents:httpcore:jar:4.3.2
- org.apache.httpcomponents:httpclient-osgi:jar:4.3.4
- org.apache.httpcomponents:httpclient:jar:4.3.4
- org.apache.httpcomponents:httpclient-cache:jar:4.3.4
- org.apache.httpcomponents:fluent-hc:jar:4.3.4
- org.apache.httpcomponents:httpmime:jar:4.3.4
- com.ibm.json:json4j:jar:1.0.0.1wl
- tagsoup:tagsoup:jar:1.2.osgi
- com.notnoop.apns:apns:jar:0.2.3
- org.apache.geronimo.specs:geronimo-jpa_1.0_spec:jar:1.1.2
- org.apache.openjpa:openjpa:jar:1.2.2
- net.sourceforge.serp:serp:jar:1.13.1.osgi
- commons-collections:commons-collections:jar:3.2.2
- org.apache.commons:commons-collections4:jar:4.1
- org.ow2.asm:asm:jar:3.2.0.v200909071300
- bouncycastle:bcprov-jdk14:jar:1.45
- com.sun.xml.bind:jaxb-impl:jar:2.1.12.osgi
- org.scannotation:scannotation:jar:1.0.2.osgi
- com.google.code.gson:gson:jar:2.3.1
- joda-time:joda-time:jar:2.3
- com.ibm.g11n:ibm-accept-lang:jar:20130708
- com.ibm:license_metric_logger:jar:2.1.1
- com.google.api-client:google-api-client:jar:1.18.0-rc
- com.google.oauth-client:google-oauth-client:jar:1.18.0-rc
- com.google.http-client:google-http-client:jar:1.18.0-rc
- com.google.code.findbugs:jsr305:jar:1.3.9
- com.google.api-client:google-api-client-gson:jar:1.18.0-rc
- com.google.http-client:google-http-client-gson:jar:1.18.0-rc
- com.ibm.mobile:cloudant-client:jar:1.0.1-wl-201506181645
- org.osgi:org.osgi.core:jar:4.3.1
- org.osgi:org.osgi.compendium:jar:4.2.0

- org.mozilla:rhino:jar:1.7R4
- javax.transaction:jta:jar:1.1.1.v201105210645
- javax.mail:mail:jar:1.4.0.v201005080615
- com.ibm.web20fep.jaxrs:ibm_web20_jsr311-api:jar:1.1
- com.ibm.web20fep.jaxrs:ibm_web20_wink_no_abdera_no_jackson:jar:1.1.0.0-20110422
- com.google.guava:guava:jar:14.0.1

isSuccessful is a reserved JSON object key

isSuccessful is a reserved key name in the JSON object that is used by the MobileFirst server-client protocol. You cannot use isSuccessful as a key name in the JSON object that is returned in any of your adapter procedures.

Application authenticity

Extended application authenticity for iOS

Extended application-authenticity validation is not supported for iOS apps that are installed from the Apple app store. You can still use basic application-authenticity validation for such apps. For more information, see Note About Application Authenticity Failures in Applications Downloaded from the Apple App Store. For information about the different types of MobileFirst application-authenticity validations, see “MobileFirst application authenticity overview” on page 8-221.

Application Center requires MobileFirst Studio for importing and building the IBMAppCenter project

MobileFirst Studio is not part of IBM MobileFirst Platform Foundation for iOS, but if you purchased this product, you are entitled to the full cross-platform version of the product as well. You can install MobileFirst Studio from the Eclipse Marketplace, or download the full MobileFirst Platform Command Line Interface for iOS from IBM MobileFirst Developer Center to perform the build of the Application Center mobile client for iOS.

Application servers restrictions for MobileFirst Data Proxy

You cannot install MobileFirst Data Proxy on the following application servers:

- Apache Tomcat
- Versions of WebSphere Application Server Liberty profile earlier than V8.5.5.0.

Analytics page of the MobileFirst Operations Console

Response times in the Analytics page of the MobileFirst Operations Console depend on several factors, such as hardware (RAM, CPUs), quantity of accumulated analytics data, and IBM MobileFirst Platform Operational Analytics clustering. Consider testing your load before you integrate IBM MobileFirst Platform Operational Analytics into production.

Error when importing a MobileFirst Studio-compressed project on Mac OS X

When you try to import a compressed project into your workspace, the import process might stop due to a file permissions issue.

You see a FWLPL0019E error in the MobileFirst Studio console and logs. For example: `FWLPL0019E: Migrating the <projectName> project from version <x> to version <y> has failed because Cannot read file <z> after that the project will stay in closed state in the user's workspace.`

The compressed (.zip) file does not always preserve UNIX file permissions correctly. Thus, complete the following steps to edit the imported compressed file.

1. Expand the compressed file to a directory and change all of the file permissions to allow read access.
2. Compress the file again.
3. Reimport the file by using MobileFirst Studio.

You can change the file permissions in either the Finder or on the command line if you run **`chmod 755`** on all of the files.

Error when pushing an application or an adapter to a MobileFirst Server

When you push an application or an adapter to a MobileFirst Server, a message reports the following error: **Runtime '<*YourRuntime*>' is not available on this server**.

All applications and adapters are pushed to a MobileFirst Server runtime. The runtime is defined by your MobileFirst project that contains the applications and adapters. Before you push these applications and adapters to the MobileFirst Server, the associated runtime must also be deployed to the MobileFirst Server, and the server must be running. The error that you encountered indicates that the CLI can not locate the associated runtime on the target MobileFirst Server.

The CLI can not detect a given runtime on a MobileFirst Server for one or more of the following reasons:

- The MobileFirst Server is not running.
- There is no network connectivity between the system running the CLI and the MobileFirst Server.
- You have not deployed the runtime to the MobileFirst Server.

Actions:

1. Confirm that the MobileFirst Server is running.
   - Local MobileFirst Server:
     a. Run **`mfp status`**. For more information, see "**`status`**" on page 15-14.
     b. If the MobileFirst Server is not running:
        1) Change directories into the top level directory of the associated project. Run **`cd <your mfp back-end project>`**.
        2) Run **`mfp start`**.
        3) Run **`mfp status`** to ensure that the MobileFirst Server started.
   - Remote MobileFirst Server:
     a. Run the **`mfp server info`** command and identify the target remote server. Note the corresponding URL and `Name` values.
     b. Ping the host specified in the target server's URL. Do not include the port or protocol. For example: `ping remotehost.com`. If the host is unreachable or the pings timeout, contact the server administrator to have the server started and verify that there are no network connectivity issues.

   c. Assuming the server is running, run the **mfp server info <server>** command, where <server> is the Name value from the previous **mfp server info** command.
   d. If the MobileFirst Server information is displayed, the MobileFirst Server is running. Otherwise contact your server administrator to start the MobileFirst Server. For more information, see "**server info**" on page 15-13.
2. Determine the required runtime name.
   - The runtime name is the value of the **<name>** argument given at the time of creation via the **mfp create <name>** command. This is also the default top level project directory name of your project.
3. If you know that the MobileFirst Server is running, confirm that the required runtime is installed and running on that MobileFirst Server. For both a local and remote server:
   a. Run the **mfp server info** command and identify the target remote server Name value. The local MobileFirst Server name is typically local.
   b. Run the **mfp server info <server>** command, where <server> is the Name value from the previous step.
   c. Look at the command output and find the runtimes listed under the Runtime section.
   d. If the runtime name from Step 2 is not included, proceed to step 4.
4. Build, install and start a runtime on the MobileFirst Server.
   - Local MobileFirst Server:
     a. Change directories into the associated project's root directory. Run **cd <your mfp back-end project>**.
     b. Confirm that the local MobileFirst Server is started by running the **mfp status** command. If the server is not running, run **mfp start**.
     c. Run **mfp push local**. This builds the runtime WAR file, and automatically installs it on the local MobileFirst Server. For more information, see "**push**" on page 15-9.
     d. Refer to step 3 to ensure that the runtime is correctly installed and running on the local MobileFirst Server.
     e. If the runtime is not present, run **mfp restart** to restart the local MobileFirst Server. For more information, see "**restart**" on page 15-10.
   - Remote MobileFirst Server:
     a. Change directories into the associated project's root directory. Run **cd <your mfp back-end project>**.
     b. Run the **mfp push --nosend** command to build the runtime WAR file, but not install it. The WAR file can only be deployed on a local server. For more information, see "**push**" on page 15-9.
     c. Complete the instructions here: "Deploying MobileFirst projects" on page 11-1.
     d. Refer to step 3 to ensure that the runtime is correctly installed and running on the target remote MobileFirst Server.

FileNotFoundExceptions when file paths include spaces

The MobileFirst Development Server (an instance of the WebSphere Application Server Liberty profile server) cannot handle an Eclipse workspace path with white space. As a result, a simple app cannot be deployed or previewed. In MobileFirst Operations Console, an error message is displayed:

```
Server error. Contact the server administrator.
```

In the log file, the following error messages are logged:

```
[12/11/14 10:27:57:376 IST] 0000002f m.ibm.ws.container.service.app.deploy.ManifestClassPathUtils W S
[12/11/14 10:27:57:376 IST] 0000002f m.ibm.ws.container.service.app.deploy.ManifestClassPathUtils W S
[12/11/14 10:27:57:376 IST] 0000002f m.ibm.ws.container.service.app.deploy.ManifestClassPathUtils W S
[12/11/14 10:27:57:377 IST] 0000002f m.ibm.ws.container.service.app.deploy.ManifestClassPathUtils W S
[12/11/14 10:27:57:637 IST] 00000029 com.ibm.ws.webcontainer.osgi.webapp.WebGroup I SRVE0169I: Loadi
```

This is a known limitation of WebSphere Application Server Liberty profile 8.5.5.3. The error can happen on production MobileFirst Server if it is based on WebSphere Application Server Liberty profile 8.5.5.3. For more information, see PI26149: FILENOTFOUNDEXCEPTIONS WHEN FILE PATHS INCLUDE SPACES.

Do not use an Eclipse workspace path or a file path with white space.

FIPS 140-2 feature limitations

The following known limitations apply when you use the FIPS 140-2 feature in IBM MobileFirst Platform Foundation for iOS:

- 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 MobileFirst client and the MobileFirst Server.
  - For HTTPS communications, only the communications between the MobileFirst client and the MobileFirst 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.
- On iOS, this feature is supported on **`i386`**, **`armv7`**, and **`armv7s`** architectures.
- 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 Application Center client does not support the FIPS 140-2 feature.

For more information about this feature, see "FIPS 140-2 support" on page 12-125.

IBM MobileFirst Platform for iOS compatibility framework limitations

The compatibility framework has the following limitations:

- IMFClient:unregisterAuthenticationDelegateForRealm has no effect as WLClient does not support unregistering challenge handlers. The work-around is to create an empty challenge handler that does nothing and register it for the same realm name.
- IMFResurceRequest:setHTTPMethod has no effect as the HTTP method is immutable in WLResourceRequest class.
- In `[IMFClient initializeWithBackendRoute:backendGUID]`, the **`backendGUID`** parameter is ignored.

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

When you apply a fix pack or an interim fix to Application Center or MobileFirst Server, manual operations are required, and you might have to shut down your applications for some time. For more information, see "Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0" on page 7-1 or "Upgrading to

MobileFirst Server V7.1.0 in a production environment” on page 7-13.

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 MobileFirst Server on a cluster of IBM WebSphere Application Servers, Liberty profile, that you administer with a collective controller:

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

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.

Liberty server limitations

If you use the Liberty studio server on a 32-bit JDK 7, Eclipse might not start, and you might receive the following error: `Error occurred during initialization of VM. Could not reserve enough space for object heap. Error: Could not create the Java Virtual Machine. Error: A fatal exception has occurred. Program will exit.`

To fix this issue, use the 64-bit JDK with the 64-bit Eclipse and 64-bit Windows. If you use the 32-bit JDK on a 64-bit machine, you might configure JVM preferences to *mx512m* and *-Xms216m*.

Limited device count by application when the runtime database is Cloudant

The license tracking report can show the device count for 200 deployed applications only. Even in cases where more applications are deployed, the device count of these additional deployed applications cannot be displayed.

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 authentic
```

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.

Physical iOS device required for testing extended app authenticity

The testing of the extended app authenticity feature requires a physical iOS device, because an IPA cannot be installed on an iOS simulator.

Session-independent mode limitations

Session-independent mode became available starting from V7.1.0.

- Clients that were created using a version of IBM MobileFirst Platform Foundation for iOS earlier than V7.1.0 do not work with a MobileFirst Server that is operating in the new session-independent mode.
- Session-independent mode is not supported in configurations of IBM MobileFirst Platform Foundation for iOS that include any of the following components or environments:
  - Blackberry
  - Windows Phone Silverlight 8 (native)
  - Adobe AIR
  - Desktop browser
  - Mobile web apps
  - JavaME native
  - Geolocation
  - USSD

For more information, see “Session-independent mode” on page 8-108.

Support of Oracle 12c by MobileFirst Server

The installation tools of the MobileFirst Server (Installation Manager, Server Configuration Tool, and Ant tasks) support installation with Oracle 12c as a database.

The users and tables can be created by the installation tools but the database, or databases, must exist before you run the installation tools.

User Certificate Authentication feature limitations

The following known limitations apply when you use the User Certificate Authentication feature in IBM MobileFirst Platform Foundation for iOS:

- This feature is not supported by 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 MobileFirst 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 by the User Certificate Authentication feature.
- This feature cannot be used when you use the IBM HTTP Server between the client device and the MobileFirst Server. The IBM HTTP Server is unable to provide user identity to the MobileFirst Server after authenticating with the client certificate.

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 MobileFirst Server" on page 6-341
- "Troubleshooting IBM HTTP Server startup" on page 6-319
- "Troubleshooting to find the cause of installation failure" on page 6-341
- "Troubleshooting a MobileFirst Server upgrade with Derby as the database" on page 6-342
- WebSphere Application Server Liberty profile: troubleshooting Administration Services
- WebSphere Application Server full profile: troubleshooting Administration Services in a stand-alone topology
- WebSphere Application Server Network Deployment: troubleshooting Administration Services
- Apache Tomcat: troubleshooting Administration Services
- "Troubleshooting server farm configuration issues" on page 6-355
- "Troubleshooting a Cast Iron adapter – connectivity issues" on page 8-85
- "Troubleshooting JSONStore" on page 8-134
- "Troubleshooting simple data sharing" on page 8-248.
- "Troubleshooting a corrupt login page (Apache Tomcat)" on page 12-78
- "Troubleshooting failure to authenticate to Application Center and applications that use the basic registry element" on page 6-356
- "Troubleshooting push notification problems" on page 8-187
- "Troubleshooting a Request Timeout error" on page 8-4
- "Troubleshooting token licensing problems" on page 6-123

For more information about known limitations or issues in the product, and removed or deprecated features, see "Release notes" on page 3-1.

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

Tutorials, samples, and additional resources

Tutorials and samples help you get started with and learn about IBM MobileFirst Platform Foundation for iOS. Use them to evaluate what the product can do for you.

Tutorials and associated samples

For you to learn the most important features of IBM MobileFirst Platform Foundation for iOS, tutorials are available on the Getting Started page of the Developer Center for IBM MobileFirst Platform Foundation for iOS.

Tutorials are organized in categories.

Each tutorial is composed of web pages to learn the steps and one or two companion samples to practice and reuse. The samples are provided as compressed files and contain pieces of code or script files that support the step-by-step instructions. When a tutorial includes some exercises, a companion sample provides the solutions to these exercises.

The same page provides links for you to download compressed files that contain the materials for the tutorials and samples.

Sample applications

Demonstrations are available from the Starter application samples page of the Developer Center for a collection of features.

Additional documentation

The Additional documentation page of the Developer Center provides more useful links, including a guide to scalability and hardware sizing.

Terms and conditions

Before you use the IBM MobileFirst Platform Foundation for iOS Getting Started modules, exercises, and code samples that are available from Getting Started pages, you must agree on the terms and conditions that are 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.

The resources might include applicable third-party licenses. Review the third-party licenses before you use any of the resources. You can find the third-party licenses that apply to each sample in the `notices.txt` file that is included with each sample.

Installing and configuring

This topic is intended for developers and administrators who want to install and configure IBM MobileFirst Platform Foundation for iOS.

This topic describes the tasks required to install and configure the different components of IBM MobileFirst Platform Foundation for iOS. It also contains information about installing and configuring database and application server software that you need to support the runtime database.

For more information about how to size your system, see the Scalability and Hardware Sizing document and the hardware calculator spreadsheet at the Developer Center website for IBM MobileFirst Platform Foundation.

Consider compatibility with earlier releases before you decide what version of each product component to install. For more information, see "Version compatibility" on page 7-1.

Installation overview

IBM MobileFirst Platform Foundation for iOS provides the following installable components: MobileFirst Platform Command Line Interface for iOS and MobileFirst Server. This section gives an overview of the installation process.

Installing MobileFirst Platform Command Line Interface for iOS (CLI)

You install the CLI by downloading the CLI `.zip` file and decompressing it on your computer. Then, you must run the installer in the correct platform.

Installing MobileFirst Server with IBM Installation Manager

To ensure the correct installation of MobileFirst Server, see "Installation prerequisites" on page 6-4.

You must install IBM Installation Manager 1.6.3 or later separately before installing IBM MobileFirst Platform Foundation for iOS. For more information, see "Running IBM Installation Manager" on page 6-32.

Note: IBM Installation Manager is sometimes referred to as *IBM Rational Enterprise Deployment* on the eXtreme Leverage, Passport Advantage sites, and on the distribution disks. 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`.

You then use IBM Installation Manager to install MobileFirst 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 MobileFirst Server" on page 6-3.

Upgrading from earlier versions

The preceding sections provide an overview of IBM MobileFirst Platform Foundation for iOS "first time" installations. For information about upgrading existing installations of MobileFirst Server to a later version, see “Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0” on page 7-1.

Consider compatibility with earlier releases before you decide what version of each product component to install. For more information about compatibility between releases, see “Version compatibility” on page 7-1.

Installing CLI

Follow these instructions to install IBM MobileFirst Platform Command Line Interface for iOS (CLI).

Before you begin

You must have the Java Developer Kit (JDK) installed on your machine to use the CLI. Additionally, certain operating systems require certain software as a prerequisite. To generate a compatibility report, see Software Product Compatibility Reports.

You must have the *JAVA_HOME* environment variable set to your JDK directory. For example:

- Windows: `C:\Program Files\Java\jdk1.7.0_60`
- Mac OSX: `/Library/Java/JavaVirtualMachines/jdk1.7.0_67.jdk/Contents/Home`

Procedure

1. Download the IBM MobileFirst Platform Command Line Interface for iOS with the rest of IBM MobileFirst Platform Foundation for iOS from IBM Passport Advantage.
2. In the Finder, double-click the `mobilefirst_ios_cli_installer_7.0.0.zip` package. The CLI is packaged as a single compressed file, which contains installation executable files for each platform, as listed:
   - Readme file
   - `install_mac.app`
   - `resources/`
3. From the Finder, right-click the **install_mac.app** file and select **Open**. A GUI appears which guides you through the installation of IBM MobileFirst Platform Command Line Interface for iOS. Follow the instructions to complete your installation.
4. On completion of your installation, log out and then log back in. This action ensures that the **`mobilefirst`** and **`mfp`** commands are on your system path.

Uninstalling command-line tools for developers

Follow these instructions to uninstall the IBM MobileFirst Platform Command Line Interface for iOS.

Before you begin

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

Procedure

GUI Uninstallation: Select and run the uninstall. A GUI appears which guides you through the uninstallation of the IBM MobileFirst Platform Command Line Interface for iOS. Follow the instructions to complete your uninstallation.

Installing MobileFirst 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 MobileFirst Platform Foundation for iOS.

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 MobileFirst Server installer copies onto your computer all the tools and libraries that are required for deploying a MobileFirst project or the IBM MobileFirst Platform Application Center in production, and IBM SmartCloud® Analytics Embedded.

MobileFirst 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 MobileFirst Server installer.

The installer can also help with upgrading an existing installation of MobileFirst Server to the current version. See "Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0" on page 7-1.

Before you install, take the time to consider the server topology in which you will deploy the administration components and the runtimes. The supported topologies are introduced in "Planning deployment of administration components and runtimes" on page 6-8.

The following topics describe the installation of MobileFirst Server, installation prerequisites, and the procedures for a manual installation and configuration of Application Center. After MobileFirst Server is installed, a MobileFirst project must be deployed to an application server. This deployment installs a IBM MobileFirst Platform Operations Console that can be used to upload applications and adapters. The instructions in "Tutorial for a basic installation of MobileFirst Server" on page 6-24 are based on a simple installation scenario. For a complete description of the process of deploying a MobileFirst project, see "Deploying MobileFirst projects" on page 11-1.

Planning the installation of MobileFirst 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 MobileFirst 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 6-127.
  - 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 MobileFirst Operations 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 MobileFirst 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" on page 2-9.

Important: If a version of MobileFirst Server is already installed, review "Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0" on page 7-1 before you install MobileFirst Server and deploy a MobileFirst project on the same application server or databases. Failure to do so can result in an incomplete installation and a non-functional MobileFirst Server.

Download the IBM MobileFirst Platform Foundation for iOS package from IBM Passport Advantage.

Ensure that you have the latest fix packs for the IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Server installation.

MobileFirst 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.
- IBM Cloudant

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 MobileFirst Server.

The MobileFirst installer can install the IBM MobileFirst Platform 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 MobileFirst 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 MobileFirst Server installer. However, you need them before you deploy IBM MobileFirst Platform Foundation for iOS projects.

The IBM MobileFirst Platform Foundation for iOS 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
- IBM MobileFirst Platform Cloudant Data Layer Local Edition (on Linux)

File system prerequisites

To install IBM MobileFirst Platform Foundation for iOS to an application server, the MobileFirst 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 MobileFirst 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 MobileFirst installation tools.

Introduction to the MobileFirst Server components

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

MobileFirst Server components run as web applications on an application server. To set up MobileFirst Server, you must first choose which of the following supported application servers you want to use:
- WebSphere Application Server
- WebSphere Application Server Liberty
- Apache Tomcat

For implementations that require several servers, you might want to set up one of the following topologies:
- 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.)
- A farm of individual application servers. Specific installation steps must be taken. See "Installing a server farm" on page 6-127.

The following sections describe each of the following MobileFirst Server components:
- "MobileFirst runtime environments."
- "IBM MobileFirst Platform Operations Console and Administration Services" on page 6-7
- "IBM MobileFirst Platform Application Center" on page 6-7
- " Operational Analytics" on page 6-8

MobileFirst runtime environments

The MobileFirst 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 IBM MobileFirst Platform Command Line Interface for iOS. Each runtime environment can host one or more MobileFirst 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 unless you implement a disaster recovery topology with replication and conflict management as described in Active/Active topology for the MobileFirst Platform Foundation Server.

IBM MobileFirst Platform Operations Console and Administration Services

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

- The MobileFirst Operations Console application.
- The MobileFirst Administration Services application.

The MobileFirst Operations 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 MobileFirst Operations Console is supported by the MobileFirst Administration Services application. This application acts as host for all the REST services and administration tasks. Both the MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst 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 MobileFirst Administration Services, so methods in this MBean are private and are not meant to be accessed directly.

A database is required for the MobileFirst Operations Console and Administration Services.

IBM MobileFirst Platform 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 MobileFirst 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 12-65.

Operational Analytics

MobileFirst 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 "Operational analytics" on page 13-6.

Planning deployment of administration components and runtimes

Plan the deployment of MobileFirst administration components and runtimes depending on the server topology that you use.

See "Introduction to the MobileFirst Server components" on page 6-6 for an introduction to the administration components and runtimes.

You can use the following topologies of application servers:

- Stand-alone server: WebSphere Application Server Liberty profile, Apache Tomcat, or WebSphere Application Server full profile
- Server farm: WebSphere Application Server Liberty profile, Apache Tomcat, or WebSphere Application Server full profile
- WebSphere Application Server Network Deployment cell

Depending on the application server topology that you use, you can deploy either symmetrically or asymmetrically. In symmetrical deployment, you must install runtimes and administration components on the same application server. In asymmetric deployment, you can install the runtimes on different application servers from the administration components.

Modes of deployment

Two modes are available to deploy the MobileFirst administration components and the runtimes in the application server infrastructure.

- Symmetric deployment: the MobileFirst administration components (MobileFirst Operations Console and administration service applications) are deployed in the same Java Virtual Machine (JVM) as the runtimes.
- Asymmetric deployment: The administration components are deployed in a different JVM from the runtimes.

Asymmetric deployment is only supported for WebSphere Application Server Network Deployment cell topology.

Stand-alone server topology:

You can configure a stand-alone topology for WebSphere Application Server full profile, WebSphere Application Server Liberty profile, and Apache Tomcat.

In this topology, all the administration components and the runtimes are deployed in a single Java Virtual Machine (JVM).

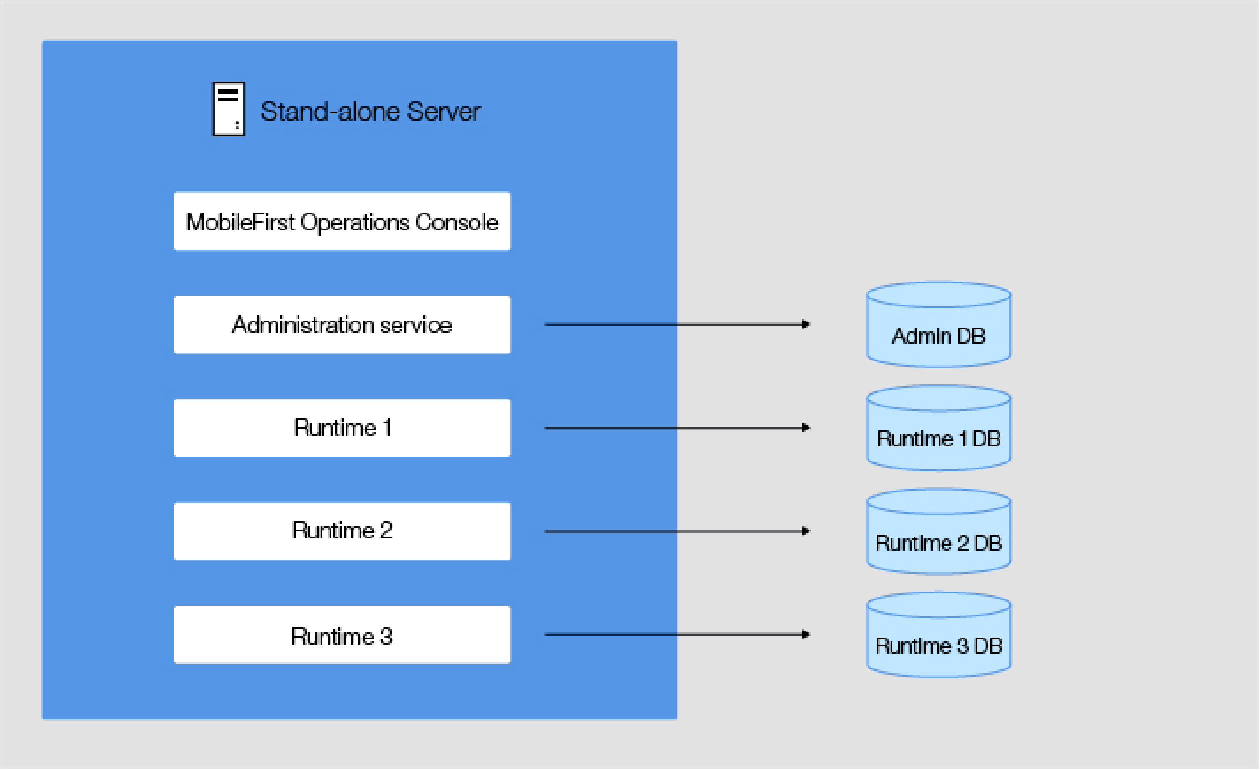


Figure 6-1. Topology of a stand-alone server

With one JVM, only symmetric deployment is possible with the following characteristics:

- One or several administration components can be deployed. Each MobileFirst Operations Console communicates with one administration service.
- One or several runtimes can be deployed.
- One MobileFirst Operations Console can manage several runtimes.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema.
- Each runtime uses its own runtime database schema.

Configuration of JNDI properties

Some JNDI properties are required to enable Java Management Extensions (JMX) communication between the administration service and the runtime, and to define the administration service that manages a runtime. For details about these properties, see “List of JNDI properties for MobileFirst Server administration” on page 6-101 and “JNDI environment entries for MobileFirst projects in production” on page 11-67.

Stand-alone WebSphere Application Server Liberty profile server

The following global JNDI properties are required for the administration services and for the runtimes.

Table 6-1. Global JNDI properties for administration services and runtimes in WebSphere Application Server Liberty stand-alone topology

JNDI properties	Values
ibm.worklight.topology.platform	"Liberty"
ibm.worklight.topology.clustermode	"Standalone"
ibm.worklight.admin.jmx.host	Hostname of the WebSphere Application Server Liberty profile server
ibm.worklight.admin.jmx.port	Port of the REST connector that is the port of the **httpsPort** attribute declared in the `<httpEndpoint>` element of the `server.xml` file of WebSphere Application Server Liberty profile server. This property has no default value.
ibm.worklight.admin.jmx.user	The user name of the WebSphere Application Server Liberty administrator, which must be identical to the name defined in the `<administrator-role>` element of the `server.xml` file of the WebSphere Application Server Liberty profile server.
ibm.worklight.admin.jmx.pwd	Password of the WebSphere Application Server Liberty administrator user.

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

Stand-alone Apache Tomcat server

The following local JNDI properties are required for the administration services and for the runtimes.

Table 6-2. Local JNDI properties for administration services and runtimes in Apache Tomcat stand-alone topology

JNDI properties	Values
ibm.worklight.topology.platform	"Tomcat"
ibm.worklight.topology.clustermode	"Standalone"

JVM properties are also required to define Java Management Extensions (JMX) Remote Method Invocation (RMI). For more information, see "Configuring Apache Tomcat" on page 6-54.

If the Apache Tomcat server is running behind a firewall, the **ibm.worklight.admin.rmi.registryPort** and

ibm.worklight.admin.rmi.serverPort JNDI properties are required for the administration service. See "Configuring Apache Tomcat" on page 6-54.

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

Stand-alone WebSphere Application Server

The following local JNDI properties are required for the administration services and for the runtimes.

Table 6-3. Local JNDI properties for administration services and runtimes in WebSphere Application Server stand-alone topology

JNDI properties	Values
ibm.worklight.topology.platform	"WAS"
ibm.worklight.topology.clustermode	"Standalone"
ibm.worklight.admin.jmx.connector	The JMX connector type; the value can be "SOAP"or "RMI".

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

Server farm topology:

You can configure a farm of WebSphere Application Server full profile, WebSphere Application Server Liberty profile, or Apache Tomcat application servers.

A farm is a set of individual servers where the same components are deployed and where the same administration database and runtime database are shared between the servers. The farm topology enables the load of MobileFirst applications to be distributed across several servers. Each server in the farm must be a Java Virtual Machine (JVM) of the same type of application server; that is, a homogeneous server farm.

In this topology, all the administration components and the runtimes are deployed on every server in the farm.

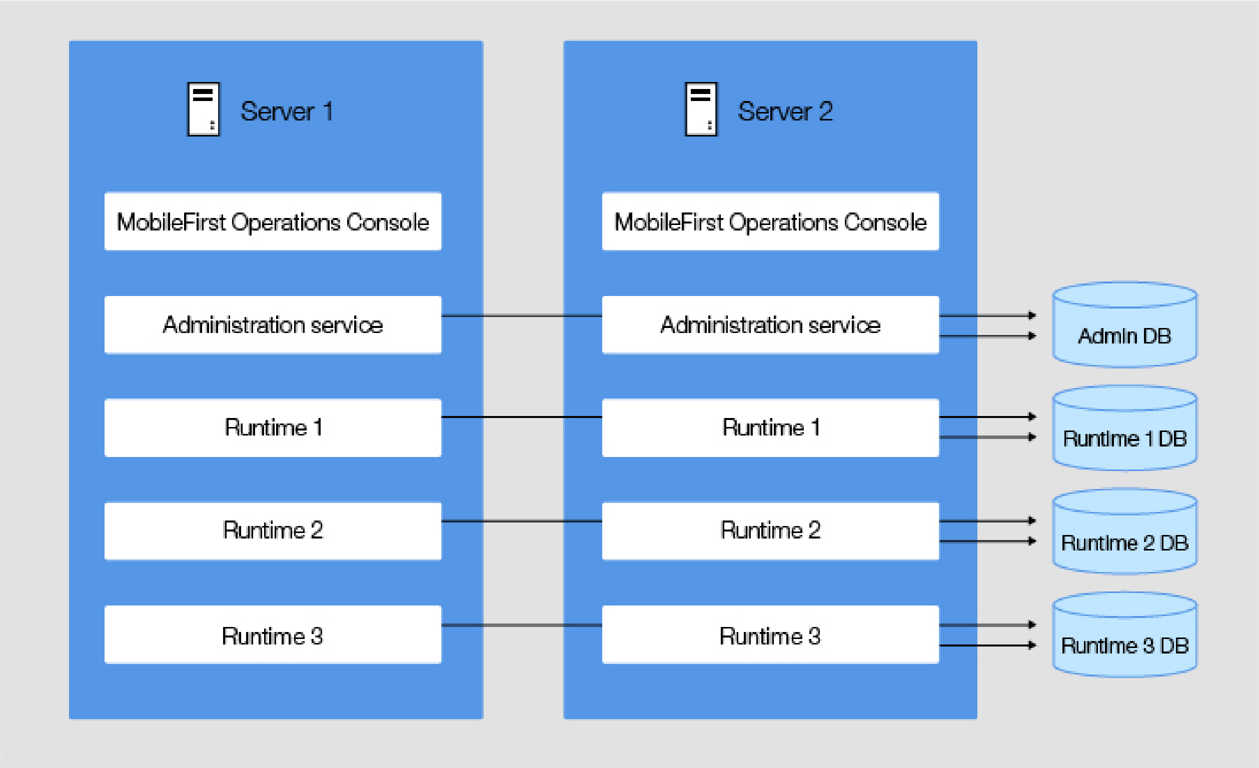


Figure 6-2. Topology of a server farm

This topology supports only symmetric deployment. The runtimes and the administration components must be deployed on every server in the farm. The deployment of this topology has the following characteristics:

- One or several administration components can be deployed. Each instance of MobileFirst Operations Console communicates with one administration service.
- The administration components must be deployed on all servers in the farm.
- One or several runtimes can be deployed.
- The runtimes must be deployed on all servers in the farm.
- One MobileFirst Operations Console can manage several runtimes.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema. All deployed instances of the same administration service share the same administration database schema.
- Each runtime uses its own runtime database schema. All deployed instances of the same runtime share the same runtime database schema.

Configuration of JNDI properties

Some JNDI properties are required to enable JMX communication between the administration service and the runtime of the same server, and to define the administration service that manages a runtime. For convenience, the tables in this section list these properties. For instructions about how to install a server farm, see “Installing a server farm” on page 6-127. For details about the JNDI properties, see “List of JNDI properties for MobileFirst Server administration” on page 6-101 and “JNDI environment entries for MobileFirst projects in production” on page 11-67.

WebSphere Application Server Liberty profile server farm

The following global JNDI properties are required in each server of the farm for the administration services and for the runtimes.

Table 6-4. Global JNDI properties for administration services and runtimes in server farm topology of WebSphere Application Server Liberty profile

JNDI properties	Values
ibm.worklight.topology.platform	"Liberty"
ibm.worklight.topology.clustermode	"Farm"
ibm.worklight.admin.jmx.host	Hostname of the WebSphere Application Server Liberty profile server
ibm.worklight.admin.jmx.port	Port of the REST connector that must be identical to the value of the **httpsPort** attribute declared in the `<httpEndpoint>` element of the `server.xml` file of the WebSphere Application Server Liberty profile server. `<httpEndpoint id="defaultHttpEndpoint" httpPort="9080" httpsPort="9443" host="*" />`

ibm.worklight.admin.jmx.user	The user name of the WebSphere Application Server Liberty administrator that is defined in the `<administrator-role>` element of the `server.xml` file of the WebSphere Application Server Liberty profile server.
	`<administrator-role>`
	`  <user>WorklightRESTUser</user>`
	`</administrator-role>`

ibm.worklight.admin.jmx.pwd	Password of the WebSphere Application Server Liberty administrator user.

The **ibm.worklight.admin.serverid** JNDI property is required for the administration service to manage the server farm configuration. Its value is the server identifier, which must be different for each server in the farm.

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

Apache Tomcat server farm

The following global JNDI properties are required in each server of the farm for the administration services and for the runtimes.

Table 6-5. Global JNDI properties for administration services and runtimes in server farm topology of Apache Tomcat

JNDI properties	Values
ibm.worklight.topology.platform	"Tomcat"
ibm.worklight.topology.clustermode	"Farm"

JVM properties are also required to define Java Management Extensions (JMX) Remote Method Invocation (RMI). For more information, see “Configuring Apache Tomcat” on page 6-54.

The **ibm.worklight.admin.serverid** JNDI property is required for the administration service to manage the server farm configuration. Its value is he server identifier, which must be different for each server in the farm.

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

WebSphere Application Server full profile server farm

The following global JNDI properties are required on each server in the farm for the administration services and for the runtimes.

Table 6-6. Global JNDI properties for administration services and runtimes in server farm topology of WebSphere Application Server full profile

JNDI properties	Values
ibm.worklight.topology.platform	“WAS”
ibm.worklight.topology.clustermode	“Farm”
ibm.worklight.admin.jmx.connector	“SOAP”

The following JNDI properties are required for the administration service to manage the server farm configuration.

JNDI properties	Values
ibm.worklight.admin.jmx.user	The user name of WebSphere Application Server. This user must be defined in the WebSphere Application Server user registry.
ibm.worklight.admin.jmx.pwd	The password of the WebSphere Application Server user.
ibm.worklight.admin.serverid	The server identifier, which must be different for each server in the farm and identical to the value of this property used for this server in the server farm configuration file.

Several administration components can be deployed to enable the same JVM to run on separate administration components that manage different runtimes.

When you deploy several administration components, you must specify the following values:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.

- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** JNDI property as the value defined for the administration service that manages the runtime.

WebSphere Application Server Network Deployment topologies:

The administration components and the runtimes are deployed in servers or clusters of the WebSphere Application Server Network Deployment cell.

Examples of these topologies support either asymmetric or symmetric deployment, or both. You can, for example, deploy the administration components in one cluster and the runtimes managed by these components in another cluster.

Symmetric deployment in the same server or cluster

Figure 6-3 shows symmetric deployment where the runtimes and the administration components are deployed in the same server or cluster.

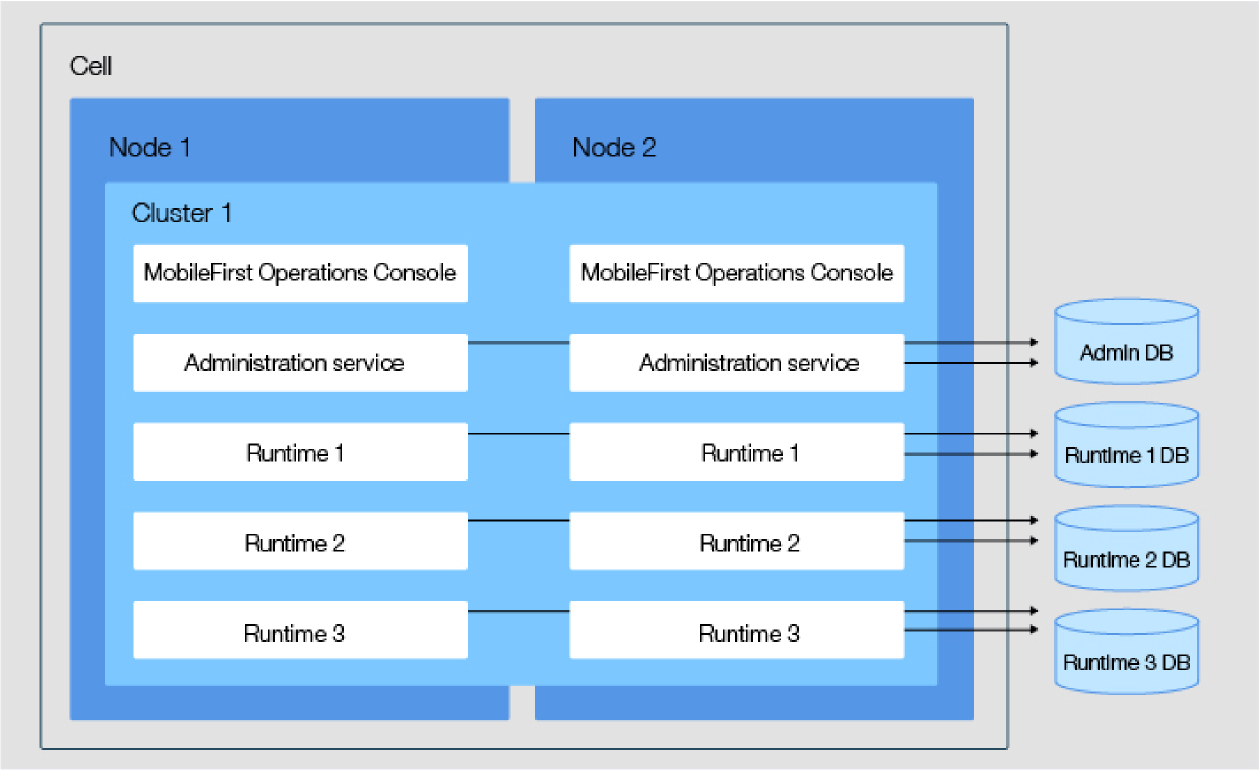


Figure 6-3. Symmetric deployment, same server or cluster

The deployment of this topology has the following characteristics:

- One or several administration components can be deployed in one or several servers or clusters of the cell. Each instance of MobileFirst Operations Console communicates with one administration service.
- One or several runtimes can be deployed in the same server or cluster as the administration components that manage them.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema.
- Each runtime uses its own runtime database schema.

Asymmetric deployment with runtimes and administration services in different server or cluster

Figure 6-4 shows a topology where the runtimes are deployed in a different server or cluster from the administration services.

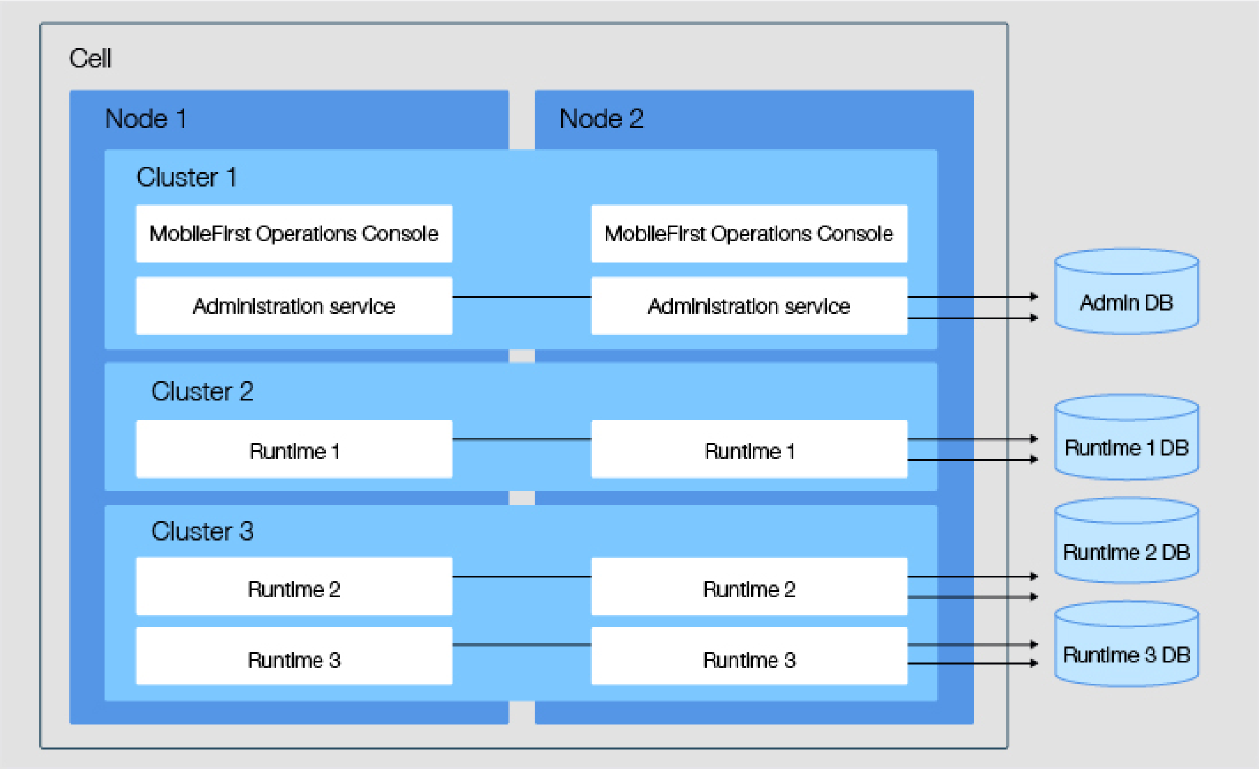


Figure 6-4. Asymmetric deployment, different server or cluster

The deployment of this topology has the following characteristics:

- One or several administration components can be deployed in one or several servers or clusters of the cell. Each instance of MobileFirst Operations Console communicates with one administration service.
- One or several runtimes can be deployed in other servers or clusters of the cell.
- One MobileFirst Operations Console manages several runtimes deployed in the other servers or clusters of the cell.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema.
- Each runtime uses its own runtime database schema.

This topology is advantageous, because it enables the runtimes to be isolated from the administration components and from other runtimes. It can be used to provide performance isolation, to isolate critical applications, and to enforce Service Level Agreement (SLA).

Symmetric and asymmetric deployment

Figure 6-5 on page 6-17 shows an example of symmetric deployment in Cluster1 and of asymmetric deployment in Cluster2, where Runtime2 and Runtime3 are deployed in a different cluster from the administration components. MobileFirst

Operations Console manages the runtimes deployed in Cluster1 and Cluster2.

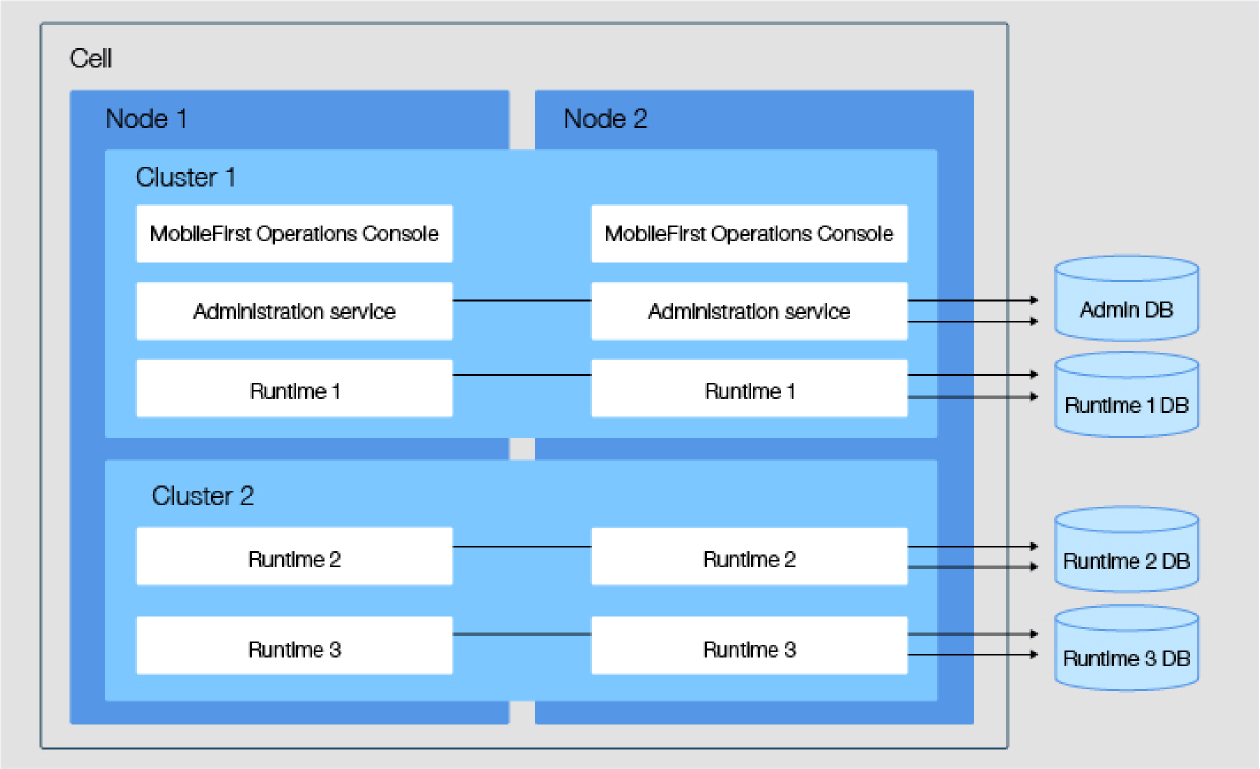


Figure 6-5. Symmetric and asymmetric deployment in different clusters of a cell

The deployment of this topology has the following characteristics:

- One or several administration components can be deployed in one or several servers or clusters of the cell. Each instance of MobileFirst Operations Console communicates with one administration service.
- One or several runtimes can be deployed in one or several servers or clusters of the cell.
- One MobileFirst Operations Console can manage several runtimes deployed in the same or other servers or clusters of the cell.
- One runtime is managed by only one MobileFirst Operations Console.
- Each administration service uses its own administration database schema.
- Each runtime uses its own runtime database schema.

Configuration of JNDI properties

Some JNDI properties are required to enable JMX communication between the administration service and the runtime, and to define the administration service that manages a runtime. For details about these properties, see “List of JNDI properties for MobileFirst Server administration” on page 6-101 and “JNDI environment entries for MobileFirst projects in production” on page 11-67.

The following local JNDI properties are required for the administration services and for the runtimes:

Table 6-7. Local JNDI properties for administration services and runtimes in WebSphere Application Server Network Deployment topologies.

JNDI properties	Values
ibm.worklight.topology.platform	WAS
ibm.worklight.topology.clustermode	Cluster
ibm.worklight.admin.jmx.connector	The JMX connector type to connect with the Deployment Manager. The value can be SOAP or RMI. SOAP is the default and preferred value. Use RMI if the SOAP port is disabled.
ibm.worklight.admin.jmx.dmgr.host	The host name of the Deployment Manager.
ibm.worklight.admin.jmx.dmgr.port	The RMI or the SOAP port used by the Deployment Manager, depending on the value of **ibm.worklight.admin.jmx.connector**.

Several administration components can be deployed to enable you to run the same server or cluster with separate administration components managing each of the different runtimes.W hen several administration components are deployed, you must specify:

- On each administration service, a unique value for the local **ibm.worklight.admin.environmentid** JNDI property.
- On each runtime, the same value for the local **ibm.worklight.admin.environmentid** as the value defined for the administration service that manages that runtime.

If the virtual host that is mapped to an administration services application is not the default host, you must set the following properties on the administration services application:

- **ibm.worklight.admin.jmx.user**: the user name of the WebSphere Application Server administrator
- **ibm.worklight.admin.jmx.pwd**: the password of the WebSphere Application Server administrator

Using a reverse proxy with server farm and WebSphere Application Server Network Deployment topologies:

You can use a reverse proxy with distributed topologies. If your topology uses a reverse proxy, configure the required JNDI properties for the administration service.

See the Glossary for the definition of a reverse proxy.

You can use a reverse proxy, such as IBM HTTP Server, to front server farm or WebSphere Application Server Network Deployment topologies. In this case, you must configure the administration components appropriately.

You can call the reverse proxy from:

- The browser when you access MobileFirst Operations Console.
- The runtime when it calls the administration service.
- The MobileFirst Operations Console component when it calls the Administration services.

If the reverse proxy is in a DMZ (a firewall configuration for securing local area networks) and a firewall is used between the DMZ and the internal network, this firewall must authorize all incoming requests from the application servers.

When a reverse proxy is used in front of the application server infrastructure, the following JNDI properties must be defined for the administration service.

Table 6-8. JNDI properties for reverse proxy.

JNDI properties	Values
ibm.worklight.admin.proxy.protocol	The protocol used to communicate with the reverse proxy, which can be HTTP or HTTPS.
ibm.worklight.admin.proxy.host	The host name of the reverse proxy.
ibm.worklight.admin.proxy.port	The port number of the reverse proxy.

The **ibm.worklight.admin.endpoint** property that references the URL of the reverse proxy is also required for MobileFirst Operations Console. See “Defining the endpoint of the MobileFirst Administration services” on page 6-91.

For detailed instructions to configure an IBM HTTP Server or a Data Store, see “Typical topologies of a MobileFirst instance in an extranet infrastructure” on page 6-299.

Related information

- “Integration and authentication with a reverse proxy” on page 14-3

Planning the creation of the databases

You must plan the creation of the databases that are needed for the Administration Services, the MobileFirst runtime environments, and the Application Center Services, if you choose to install Application Center.

The installation of the MobileFirst Server requires the following databases:

- For the Administration Services, an administration database.
- For each MobileFirst runtime environment, a runtime database.

Note: By default, the relational databases have the following names and kind attributes, as defined in table 1 of “Ant **configuredatabase** task reference” on page 15-14:

- The default name of the administration database is WLADMIN, and its kind is WorklightAdmin.
- The default name of the runtime database is WRKLGHT, and its kind is Worklight.

By default, the Cloudant databases have the following names, and kind attributes:

- The default name of the administration database is mfp_admin_db, and its kind is WorklightAdmin.
- The default name of the runtime database is in the form *prefix_contextroot*_runtime_db, where *prefix* is an optional prefix, and *contextroot* is the context root of your MobileFirst runtime, without the leading slash, and the database kind is Worklight.

Optionally, the Application Center can be installed. The Application Center also requires a database. Its database kind is **ApplicationCenter**, as defined in Table 15-1 on page 15-15 of the page "Ant **configuredatabase** task reference" on page 15-14.

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

The relational 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 relational database beforehand. For more information about the creation of these databases, see "Optional creation of the administration database" on page 6-47. For more information about the MobileFirst runtime environments, see "Optional creation of databases" on page 11-6.

For DB2, the administration, and the runtime 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 relational 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 IBM MobileFirst Platform 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 MobileFirst administration and runtime components. The user credentials appear in the application server configuration.

MobileFirst Server data is stored in the databases that are described in "Introduction to the MobileFirst Server components" on page 6-6. 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 MobileFirst 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 MobileFirst runtime environment from the database of the MobileFirst administration component, improves security. These database users have no relation to the standard MobileFirst Server groups. The following table shows the minimal permissions that the database administrator must define on the MobileFirst Server databases for these users:

Table 6-9. Minimal permissions defined by the database administrator.

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

Table 6-9. Minimal permissions defined by the database administrator (continued).

Database permission	Use MobileFirst 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

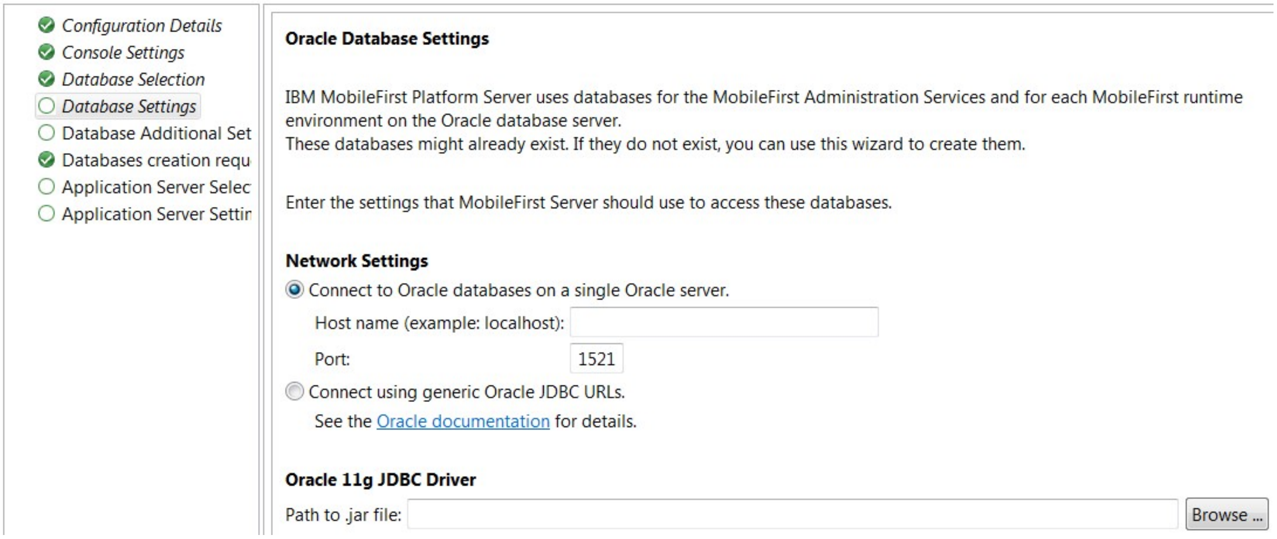
These minimal permissions also apply to the database user of the (optional) Application Center database.

Using complex Oracle connection descriptors:

For some topologies of the Oracle DBMS, 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 MobileFirst Server administration, and the MobileFirst 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 6-212.
   - For installing the MobileFirst Server administration, see "Creating the Oracle database for MobileFirst Server administration" on page 6-50.
   - For deploying the MobileFirst project WAR file, see "Creating the Oracle database" on page 11-8.
2. In IBM Installation Manager, or in the Server Configuration Tool, you must use a generic Oracle JDBC URL instead of the host name and port.


✓ Configuration Details
✓ Console Settings
✓ Database Selection
○ Database Settings
○ Database Additional Set
✓ Databases creation requ
○ Application Server Selec
○ Application Server Settir

Oracle Database Settings

IBM MobileFirst Platform Server uses databases for the MobileFirst Administration Services and for each MobileFirst runtime environment on the Oracle database server.
These databases might already exist. If they do not exist, you can use this wizard to create them.

Enter the settings that MobileFirst Server should use to access these databases.

Network Settings

● Connect to Oracle databases on a single Oracle server.

Host name (example: localhost):

Port: 1521

○ Connect using generic Oracle JDBC URLs.

See the Oracle documentation for details.

Oracle 11g JDBC Driver

Path to .jar file:

Figure 6-6. 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 15-14, table 19.

   Note: The example files in “Sample configuration files” on page 15-66 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])`.

Configuring DB2 HADR seamless failover for MobileFirst 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.

Note: DB2 HADR failover is not supported for Apache Tomcat.

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 MobileFirst 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 MobileFirst 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 MobileFirst 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.

For more information about how to configure a connection to an HADR-enabled DB2 database, see Setting up a connection to an HADR-enabled DB2 database.

Planning for the use of token licensing

If the token licensing is purchased for MobileFirst Server, you have extra steps to consider in the installation planning.

Technical restrictions

Here are the technical restrictions for the use of token licensing:

Supported Platforms:

The list of platforms that support token licensing is listed at "Limitations of supported platforms for token licensing" on page 6-122. The MobileFirst Server running on a platform that is not listed might not be possible to install and configure for token licensing. The native libraries for the Rational Common Licensing client might not available for the platform or not supported.

Supported Topologies:

From the list of topologies that are listed at "Planning deployment of administration components and runtimes" on page 6-8, the non-symmetric deployments are not supported. The number of tokens that are used by a MobileFirst cluster might be incorrect. In particular, the asymmetric deployment with runtimes and administration services in different server or cluster is not supported on WebSphere Application Server Network Deployment.

Network requirement

MobileFirst Server must be able to communicate with the Rational License Key Server.

This communication requires the access to the following two ports of the license server:

- License manager daemon (**lmgrd**) port - the default port number is 27000.
- Vendor daemon (**ibmratl**) port

To configure the ports so that they use static values, see How to serve a license key to client machines through a firewall.

Installation Process

You need to activate token licensing when you run the IBM Installation Manager at installation time. For more information about the instructions for enabling token licensing, see "Installation overview for token licensing" on page 6-115.

After MobileFirst Server is installed, you must manually configure the server for token licensing. For more information, see "Installing and configuring for token licensing" on page 6-115.

The MobileFirst Server is not functional before you complete this manual configuration. In "Installing and configuring for token licensing" on page 6-115, the Rational Common Licensing client library is to be installed in your application server, and you define the location of the Rational License Key Server.

Operations

The tokens are checked out per application. The tokens are checked out when an application is uploaded, a runtime server starts, or a token is refreshed after its expiration date. At server starts, if IBM MobileFirst Platform Foundation for iOS fails to obtain the tokens it needs, the server will not be functional. If IBM MobileFirst Platform Foundation for iOS fails to renew its expired tokens during operations, the applications are deactivated. The mobile app users can no longer access the applications.

If you need to test a non-production application on a production server with token licensing enabled, you can declare the application as non-production. For more information about declaring the application type, see `<licenseAppType>` element in "Application descriptor of native API applications" on page 8-7 and also "Token licensing: setting the license app type" on page 8-21.

For more information about the retrieval of tokens during operations, see "Token license validation" on page 13-133.

Development

When you develop your application, you must declare the type of application and the license entitlement it matches (such as `NON_PRODUCTION`, `APPLICATION`, or `ADDITIONAL_BRAND_DEPLOYMENT`). For more information, see `<licenseAppType>` element in "Application descriptor of native API applications" on page 8-7 and also "Token licensing: setting the license app type" on page 8-21.

Tutorial for a basic installation of MobileFirst Server

Learn about the MobileFirst Server installation process by walking through a simple configuration, which creates a functional MobileFirst Server for demonstration purposes or tests.

Before you begin

1. Install IBM MobileFirst Platform Command Line Interface for iOS on your computer, if you have not already done so.

2. Use MobileFirst Platform Command Line Interface for iOS to create a project, which you can then run on MobileFirst Server.

About this task

This tutorial uses a simple configuration to show how to install MobileFirst Server. It is designed as an overview, to show you where to find the following tools and information:

- Tools to install a MobileFirst Server and the Application Center, and tools to deploy a MobileFirst project.
- Information about configuring MobileFirst Server and the Application Center.
- Information about manual MobileFirst Server installation.

  Note: 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 tutorial goes through the following steps:

1. "Installing WebSphere Application Server Liberty Core"
2. "Creating a server for Liberty" on page 6-26
3. "Installing the database management system" on page 6-26
4. "Installing MobileFirst Server" on page 6-27
5. "Exploring Application Center" on page 6-28
6. "Installing the MobileFirst Server administration components: Administration Services and MobileFirst Operations Console" on page 6-29
7. "Creating a simple MobileFirst project" on page 6-30
8. "Deploying a MobileFirst runtime environment with the Server Configuration Tool" on page 6-30
9. "Restarting the Liberty server and opening the MobileFirst Operations Console" on page 6-31

Installing WebSphere Application Server Liberty Core

About this task

The installer for WebSphere Application Server Liberty Core is provided as part of the package for IBM MobileFirst Platform Foundation for iOS.

Procedure

1. 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 as *IBM Rational Enterprise Deployment* on the eXtreme Leverage and Passport Advantage sites, and on the distribution disks. The file names for the images take the form `IBM Rational Enterprise Deployment` *`<version number><hardware platform>`* *`<language>`*; for example, `IBM Rational Enterprise Deployment V1.6.3.1 Windows Multilingual`.

   For more information about loading repositories with IBM Installation Manager, see step 1 of "Installing MobileFirst Server" on page 6-27. 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.

2. During the installation process, take note of the installation directory of Liberty. You need this information later on in the procedure.

Creating a server for Liberty

About this task

You use this server to install the Application Center and to deploy a MobileFirst project and its console.

Procedure

1. 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`.
2. 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. Default settings are sufficient for this tutorial.

   For more information about configuring a Liberty server, read the `README.txt` file in the Liberty installation directory.

Installing the database management system

About this task

You install a database management system so that you can use this DBMS to install the Application Center and to deploy a MobileFirst project and its console.

Procedure

Proceed as follows, depending on your database system.

- If you use IBM DB2, the installer for IBM DB2 is provided as part of the package for IBM MobileFirst Platform Foundation for iOS.
  1. Run the installer and follow the instructions.
  2. On Windows, when 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 installed and running so that the MobileFirst tools can create the required databases.
     – On Windows, use the IBM Secure Shell Server for Windows or the Cygwin `openssh` package, as described at http://docs.oracle.com/cd/E25178_01/install.1111/e22624/preinstall_req_cygwin_ssh.htm

     – On UNIX, use the `sshd` daemon.
  4. Take note of the user name and password for the DB2 administrator role.
- 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 no password is requested.

  - If no password is requested for the MySQL administrator in your installation, set a password for the administrator. Follow the instructions from the MySQL documentation.
- 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.
  3. Start the `ssh` server. On Windows, you need administrator rights.
  4. In subsequent steps, you must have that Secure Shell server running.
- If you use Cloudant, you can install IBM MobileFirst Platform Cloudant Data Layer Local Edition, which is provided as part of the package for IBM MobileFirst Platform Foundation for iOS, or you can use an account at `Cloudant.com`.

  Important: Application Center does not support Cloudant.

Installing MobileFirst Server

Procedure

1. Add the MobileFirst Server repository in IBM Installation Manager:
   a. Download the **Installation Manager Repository for IBM MobileFirst Platform Server** from Passport Advantage.
   b. Extract the file on your disk.
   c. Start IBM Installation Manager.
   d. Select **File** > **Preferences**.
   e. In the Preferences dialog, click **Add Repositories**.
   f. Select the `disk1/diskTag.inf` file from the repository directory that you extracted.
   g. Click **OK** and close the Preferences dialog.
2. Load the repository for MobileFirst Server in IBM Installation Manager and install the product.
   a. In the Configuration Choice panel, select the first choice. This option installs Application Center.
   b. In that Database Choice panel, select the name of the database management system that you installed.

      Restriction: Apache Derby is not supported by the Server Configuration Tool, which is used later in this tutorial.
   c. 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 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 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 exists, provide the password of the `SYSTEM` user that is used to create the user `APPCENTER`. If the database does not exist, enter the passwords for the `SYS` and `SYSTEM` users that are created to manage the database.

d. In the **Application Server Selection** panel, select **WebSphere Application Server**.

e. In the **Application Server Configuration** panel, select the installation directory for IBM WebSphere Application Server Liberty Core that is installed in step 2.

f. Select **simpleServer** as the server name.

g. Install the product.

The files that are described in "Distribution structure of MobileFirst Server" on page 6-42 are installed on your computer.

Exploring Application Center

About this task

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.

Procedure

1. 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
     ```

2. Open the Application Center console by using the program shortcut that the installer creates: **IBM MobileFirst Platform Server** > **Application Center**.

   Alternatively, you can enter the URL into a browser window. When a Liberty server is created with default settings, the default URL for Application Center is `http://localhost:9080/appcenterconsole/`.
3. Log in to the Application Center with the demonstration account credentials: `demo/demo` as both the user name and the password.
4. Explore further by using any of the following resources:
   - See “Configuring user authentication for Application Center” on page 6-243.
   - See “Distribution structure of MobileFirst Server” on page 6-42 for a list of MobileFirst 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 6-218. 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.

Installing the MobileFirst Server administration components: Administration Services and MobileFirst Operations Console

Procedure

1. Start the Server Configuration Tool.
   - On Linux, click the desktop menu **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Windows, click **Start** > **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Mac OS X, in the Finder, double-click the file `*mf_server_install_dir*/shortcuts/configuration-tool.sh`.

     The *mf_server_install_dir* directory is where you installed MobileFirst Server. *mf_server* is the shortcut for MobileFirst Server.

   Restriction: MobileFirst Server is not supported for production use on Mac OS X.
2. Select **Create a MobileFirst Server Configuration**.
3. Name the configuration `My MobileFirst Server`.
4. Do not change the default entries in the Configuration Description panel.
5. Do not change the default entries in the Console Settings panel, except to create a shortcut file.
6. To create the `mobilefirst-console.url` shortcut file, select the **Create MobileFirst Server shortcuts** checkbox in the Console Settings panel and select a destination directory.
7. In the Database Properties panel:
   a. Select your database.

   b. Proceed as described in “Installing MobileFirst Server” on page 6-27 when you entered data to create the database for Application Center.

   Note: If you selected Cloudant, you must provide credentials to access an existing valid Cloudant account.

8. In the Application Server panel:
   a. Proceed as described in “Installing MobileFirst Server” on page 6-27 when you entered data to create the database for Application Center.
   b. Take note of the default password and login: `admin` (for both).
9. In the Analytics settings panel, make sure that the check box to connect to MobileFirst Operational Analytics is not selected.
10. When all the data is entered, click **Deploy**.
    - The log of the deployment operations is displayed in the console.
    - An **Administration - Server deployment** folder appears in the tree view under **Log Files**.
    - After the database operation is completed, an `admdatabases` log file appears in the tree view.
    - After the deployment to the application server is complete, an `admininstall` log file appears in the tree view.

Creating a simple MobileFirst project

You create a MobileFirst runtime environment.

Procedure

1. Install command-line tools for developers on your computer. See “Installing CLI” on page 6-2.
2. Use the **`Create`** command to create a MobileFirst project.
3. Assign the name `simpleProject`, and name the application `simpleApp`.
4. Use the **`Build`** command to build the application.

Deploying a MobileFirst runtime environment with the Server Configuration Tool

Procedure

1. In the Server Configuration Tool, click **Add a MobileFirst runtime environment to a configuration** on the main panel or select **Runtime environments** > **File/Add MobileFirst runtime environment**.

   Selecting **Runtime environments** makes **File/Add MobileFirst runtime environment** available for selection.
2. In the dialog box, select the **Hello MobileFirst Server** configuration that you created in “Installing the MobileFirst Server administration components: Administration Services and MobileFirst Operations Console” on page 6-29.
3. In **Enter the name of the new runtime**, enter `First Runtime`.
4. In the **MobileFirst runtime environment Configuration Description** panel, load the WAR file that you created in the previous part.
5. In the Database Properties panel, proceed as described in “Installing MobileFirst Server” on page 6-27 when you entered data to create the database for Application Center.

   The database is selected because it is already defined in the configuration.
6. 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 `databases` log file appears in the tree view, under **Runtime**.
- After the deployment to the application server is complete, an `install` log file appears in the tree view, under **Runtime**.

Restarting the Liberty server and opening the MobileFirst Operations Console

Procedure

1. Go to the Liberty installation directory and type the following command:
   - On Linux and UNIX systems:
     ```
     bin/server stop simpleServer
     ```
   - On Windows systems:
     ```
     bin\server.bat stop simpleServer
     ```
2. Restart the server by running the following command:
   - On Linux and UNIX systems:
     ```
     bin/server start simpleServer
     ```
   - On Windows systems:
     ```
     bin\server.bat start simpleServer
     ```
3. In the shortcut directory that you specified in the **MobileFirst runtime environment Configuration Description** panel of the Server Configuration Tool, proceed as follows:
   - On Linux and UNIX systems, run the `mobilefirst-console.sh` script.
   - On Windows systems, double-click the `mobilefirst-console.url` file. On Windows 7, this shortcut can appear as `mobilefirst-console`, with a file type of Internet Shortcut.

   You can see the MobileFirst Operations Console. You can log in with the default user login and password that you created in step “Installing the MobileFirst Server administration components: Administration Services and MobileFirst Operations Console” on page 6-29 (by default `admin/admin`).

What to do next

For more information about the Server Configuration Tool, see “Deploying, updating, undeploying, or upgrading MobileFirst Server by using the Server Configuration Tool” on page 11-11.

If you want to use MobileFirst Operational Analytics, see “Installing the IBM MobileFirst Platform Operational Analytics” on page 6-191.

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

- Deploy an application as described in “Deploying applications and adapters to MobileFirst Server” on page 11-84.
- Review “Administering MobileFirst applications” on page 12-1.
- Review “Deploying the project WAR file” on page 11-5.
- Review “Configuration of MobileFirst applications on the server” on page 11-50 and “Configuring a MobileFirst project in production by using JNDI environment entries” on page 11-65

- Review the options to deploy an IBM MobileFirst Platform Foundation for iOS project manually. See “Deploying a project WAR file and configuring the application server manually” on page 11-37.

Running IBM Installation Manager

IBM Installation Manager installs the IBM MobileFirst Platform Foundation for iOS files and tools on your computer.

IBM Installation Manager helps you install, update, modify, and uninstall packages on your computer. The installer for MobileFirst Server does not support rollback operations and updates from one version to another 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 MobileFirst Platform Foundation for iOS in several different modes, including single-user and multi-user installation modes.

You can also use silent installations to deploy IBM MobileFirst Platform Foundation for iOS 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 as *IBM Rational Enterprise Deployment* on the eXtreme Leverage, Passport Advantage sites, and on the distribution disks. 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 6-5, then see “Installing and configuring the Application Center” on page 6-210 and install the Application Center before you proceed to the installation of MobileFirst Operations Console. For more information, see “Installing the MobileFirst Server administration” on page 6-47.

If you plan to install the Application Center by running Ant tasks, or manually, answer **No** to the question **Install IBM Application Center**.

Single-user versus multi-user installations

You can install MobileFirst 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 MobileFirst 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 MobileFirst 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, or Apache Tomcat).
- Modify the Application Center Derby database (if Apache Derby is chosen as your database management system).

In this case, the MobileFirst 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 MobileFirst 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 MobileFirst 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: MobileFirst Server does not support the group mode of IBM Installation Manager.

Installing a new version of MobileFirst Server

Create a fresh installation of IBM MobileFirst Platform 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 MobileFirst Server from a previous release

The way that you use IBM Installation Manager to upgrade to the latest version of MobileFirst Server depends on your upgrade path.

Before you begin

Before you apply these instructions, see "Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0" on page 7-1. It describes important steps to upgrade MobileFirst 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 Worklight Server to MobileFirst Server:
     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 MobileFirst Server automatically removes a Worklight Server installation that was installed in the same directory.
   - To upgrade from MobileFirst Server to a more recent version, click **Update**.

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

With IBM Installation Manager, you can complete a command-line installation of MobileFirst 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 MobileFirst Server on a set of computers that are configured in the same way.
- You want to install MobileFirst 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 MobileFirst 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 6-38.

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 MobileFirst 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 Application Center on an application server, use the file named `install-no-appcenter.xml`. The installation of Application Center can be done later by using Ant tasks. For more information, see "Installing the Application Center with Ant tasks" on page 6-216.

   For an installation that installs Application Center, pick the sample response file from the following table, depending on your application server and database.

Table 6-10. Sample installation response files in the `Silent_Install_Sample_Files.zip` file to install the 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 6-10. Sample installation response files in the `Silent_Install_Sample_Files.zip` file to install the 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 DBMS 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 MobileFirst Server or Worklight Server that you initially installed in the particular package group:

- MobileFirst Server uses the package group **IBM MobileFirst Platform Server**.
- Worklight Server V6.x, or later, uses the package group **IBM Worklight**.
- Worklight Server V5.x uses the package group **Worklight**.

Table 6-11. Sample uninstallation response files in the `Silent_Install_Sample_Files.zip`

Initial version of MobileFirst Server	Sample file
Worklight Server V5.x	`uninstall-initially-worklightv5.xml`
Worklight Server V6.x	`uninstall-initially-worklightv6.xml`
IBM MobileFirst Platform Server V6.x or later	`uninstall-initially-mfpserver.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 7-30 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 MobileFirst 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.
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 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: `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.
5. Install MobileFirst 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 6-12. Parameters available for silent installation.

Key	When necessary	Description	Allowed values
`user.licensed.by.tokens`	Always	Activation of token licensing If you plan to use the product with the Rational License Key Server, you must activate token licensing. In this case, set `value = "true"`. If you do not plan to use the product with Rational License Key Server, set `value = "false"`. If you activate license tokens, specific configuration steps are required after you deploy the product to an application server. For more information, see "Installing and configuring for token licensing" on page 6-115.	`true` or `false`
`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 later.	`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.

Table 6-12. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
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.
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`
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.password2	${user.appserver.selection2} == was && ${user.appserver.was.profile} != Liberty	Password of WebSphere Application Server administrator, optionally encrypted in a specific way.	

Table 6-12. 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 MobileFirst 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.
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.
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.

Table 6-12. Parameters available for silent installation (continued).

Key	When necessary	Description	Allowed values
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.oracle.appcenterr .isservicename.jdbc.url	Optional	Indicates if `user.database.mysql .appcenter.dbname` is a Service name or a SID name. If the parameter is absent then `user.database.mysql .appcenter.dbname` is considered to be a SID name.	`true` (indicates a Service name) or `false` (indicates a SID 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.
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 6-12. 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.isservicename .jdbc.url	Optional	Indicates if `user.database.oracle .appcenter.dbname` is a Service name or a SID name. If the parameter is absent then `user.database.oracle .appcenter.dbname` is considered to be a SID name.	`true` (indicates a Service name) or `false` (indicates a SID name)
user.database.oracle.appcenter.jdbc.url	${user.database.selection2} == oracle, unless ${user.database.oracle.host}, ${user.database.oracle.port}, ${user.database.oracle.appcenter.dbname} are all specified	The JDBC URL of the Oracle database for Application Center.	A valid Oracle JDBC URL. Starts with "jdbc:oracle:".
user.writable.data.user	Always	The operating system user that is allowed to run the installed server.	An operating system user name, or empty.
user.writable.data.group2	Always	The operating system users group that is allowed to run the installed server.	An operating system users group name, or empty.

Distribution structure of MobileFirst Server

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

Table 6-13. Files and subdirectories in the MobileFirst Server installation directory

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

Table 6-14. Files and subdirectories in the `WorklightServer` subdirectory

Item	Description
`worklight-jee-library.jar`	The MobileFirst Server library for production. For instructions on deploying a MobileFirst project and this library to an Application Server, see "Deploying MobileFirst projects" on page 11-1. 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 MobileFirst Server. For documentation about the Ant tasks that are provided in this library, see "Deploying MobileFirst projects" on page 11-1.
`worklight-ant-builder.jar`	A set of Ant tasks that help you build projects, applications, and adapters for use in MobileFirst 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 MobileFirst Server and deploying a MobileFirst project to an application server. For instructions on how to use these Ant projects, see "Sample configuration files" on page 15-66.
`databases`	SQL scripts to be used for the manual creation of tables for MobileFirst 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 11-20.
`encrypt.bat` and `encrypt.sh`	Tools to encrypt confidential properties that are used to configure a MobileFirst Server, such as a database password or a certificate. For information about this tool, see "Storing properties in encrypted format" on page 11-61.
`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 13-113.
`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 MobileFirst Operations Console user interface web application.

Table 6-14. Files and subdirectories in the `WorklightServer` subdirectory (continued)

Item	Description
`external-server-libraries`	JAR file and manifest file of the OAuth Trust Association Interceptor (TAI) that is used to protect application resources on WebSphere Application Server or WebSphere Application Server Liberty. For more information, see “Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty” on page 8-199. JAR file of the eXtreme Scale plug-in that is used in attribute store over IBM WebSphere eXtreme Scale. For more information, see “Attribute store over IBM WebSphere eXtreme Scale” on page 8-112.
`tokensLibs`	This directory contains the native libraries for the Rational License Key Server client. It is installed if you activated token licensing from the installation wizard, or with the parameter **`user.licensed.by.tokens`** for silent installations. For more information about how to install these libraries in your application server, see “Installing and configuring for token licensing” on page 6-115.

Table 6-15. 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/IBMAppCenter`	Contains the MobileFirst project for the mobile Client. You must build this project to create the iOS version of the mobile client.
`ApplicationCenter/console/`	**`appcenterconsole.war`** The WAR file for the Application Center console user interface web application. **`applicationcenter.war`** The WAR file for the Application Center REST services web application. **`applicationcenter.ear`** The enterprise application archive (EAR) file to be deployed under IBM PureApplication® System.

Table 6-15. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

Item	Description
`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 MobileFirst Platform Foundation for iOS.

Table 6-15. Files and subdirectories in the `ApplicationCenter` subdirectory (continued)

Item	Description
ApplicationCenter/tools	**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 6-16. Files and subdirectories in the `License` subdirectory

Item	Description
Text	License for IBM MobileFirst Platform Foundation

Table 6-17. 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 “Deploying MobileFirst projects” on page 11-1.

Table 6-18. Files and subdirectories in the `Analytics` subdirectory

Item	Description
`analytics.ear`	The IBM MobileFirst Platform Operational Analytics EAR file. Contains the `analytics-service.war` file for deployment on WebSphere Application Server and WebSphere Application Server Liberty. For installation instructions, see "Installing the IBM MobileFirst Platform Operational Analytics" on page 6-191.
`analytics-ui.war`	The WAR file for the analytics console user interface web application.
`analytics-service.war`	The WAR file for the analytics REST services web application.
`configuration-samples`	Contains the sample Ant files for deploying both the analytics console user interface and the analytics REST services web applications to an application server. For more information about how to use these Ant projects, see Sample configuration files for MobileFirst Operational Analytics.

Table 6-19. Files and subdirectories in the `Datastore` subdirectory.

Item	Description
`imf-data-proxy.war`	The WAR file for the MobileFirst Data Proxy.
`configuration-samples`	Contains the sample Ant files for deploying a MobileFirst Data Proxy to an application server. For more information about how to use these Ant projects, see "Installing the MobileFirst Data Proxy with Ant tasks" on page 6-201.

Installing the MobileFirst Server administration

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

Optional creation of the administration database

If you want to activate the option to install the administration relational database when you run the Ant tasks or the Server Configuration Tool, you must have certain database access rights that entitle you to create the databases, or the users, or both, that are required by the MobileFirst Server administration.

If you have sufficient administration credentials on the relational databases, and if you enter the administrator user name and password when prompted, or use Ant files with `dba` tags, 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.

If you want to use a Cloudant database for the MobileFirst administration, ensure that you have access to a valid Cloudant account, and that a user with basic authentication credentials is defined to create the databases. Unlike relational databases, the Cloudant databases are not created during the installation process, but when the MobileFirst administration is started after it was deployed on the application server. Both the Cloudant account URL and user credentials that are mentioned above are used to create the databases.

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

Important: This step is optional if you install IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Server administration:

During the installation of IBM MobileFirst Platform Foundation for iOS, 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 IBM MobileFirst Platform Server administration" on page 6-65 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 create the database manually, you can replace the database name (here WLADMIN) and the password with a database name and password of your choice.

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 limits:

- Database names: no more than 8 characters on all platforms
- User name and passwords: no more than 8 characters for UNIX and Linux, and no more than 30 characters for Windows

Procedure

1. Create a system user named, for example, `wluser` in a DB2 admin group such as DB2USERS, by using the appropriate commands for your operating system. Give it a password, for example, `wluser`.

   If you want multiple MobileFirst Server instances 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.

3. To create the administration database, enter database manager and SQL statements similar to the following example.

   Replace the user name wluser with your own.

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

What to do next

The installation tools can create the database tables and objects for MobileFirst Server administration in a specific schema. You can then use the same database for MobileFirst Server administration and for a MobileFirst project.

- If the IMPLICIT_SCHEMA authority is granted to the user that you created in Step 1, no further action is required. This is the default in the database creation script of Step 2.
- If the user does not have the IMPLICIT_SCHEMA authority, create a SCHEMA for the administration database tables and objects.

Creating the MySQL database for MobileFirst Server administration:

During the MobileFirst installation, 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 MySQL database manually for the IBM MobileFirst Platform Server administration” on page 6-73 instead.

About this task

The installation tools 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 create the database manually, you can replace the database name (here WLADMIN) and password with a database name and password of your choice.

Attention: On UNIX, MySQL database names are case-sensitive.

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 MobileFirst Platform Foundation for iOS runs.

Creating the Oracle database for MobileFirst Server administration:

During the installation of IBM MobileFirst Platform Foundation for iOS, the installation tools can create the administration database, except for the Oracle 12c database type, or the user and schema inside an existing database.

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 IBM MobileFirst Platform Server administration" on page 6-77 instead.

About this task

The installation tools can create the database, except for the Oracle 12c database, or the user and schema inside an existing database, 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 choice.

Attention: Lowercase characters in Oracle user names can lead to unwanted results.

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 by using either Oracle Database Control or the Oracle SQLPlus command-line interpreter.
   - Using Oracle Database Control.
     a. Connect as `SYSDBA`.
     b. Go to the **Users** page and click **Server**, then **Users** in the **Security** section.
     c. Create a user, for example WLADMIN. If you want multiple MobileFirst Server instances to connect to the general-purpose database that 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 tablespace: **`USERS`**
        - Temporary tablespace: **`TEMP`**
        - Status: **`Unlocked`**

  - Add system privilege: **CREATE SESSION**
  - Add system privilege: **CREATE SEQUENCE**
  - Add system privilege: **CREATE TABLE**
  - Add quota: **Unlimited for tablespace USERS**
- 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 WLADMIN_password DEFAULT TABLESPACE USERS QUOTA UNLIMITED
  GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO WLADMIN;
  DISCONNECT;
  ```

Creating the Cloudant database for MobileFirst Server administration:

Learn how to create Cloudant databases.

General information

To use a Cloudant database, you either need an account at `cloudant.com` that uses the Cloudant database in the Cloud, or you must install IBM MobileFirst Platform Cloudant Data Layer Local Edition on your own hardware, as described in the Cloudant Data Layer Local Edition user documentation. Whether you use `cloudant.com` or Cloudant Local, the Cloudant database is accessible via http or https, and you must have the URL to access it, as well as a user name and password.

Cloudant account

To create or access a Cloudant database at `cloudant.com` for the MobileFirst Server administration, you must already have registered a valid account for `cloudant.com`. The account registration provides you with the following information:

- The account name: The Cloudant account name is displayed in the URL. For example, the account *exampleaccount* corresponds to the URL `https://`*`exampleaccount`*`.cloudant.com`.

  Important: You must always use the https protocol, and not the http protocol, when accessing `https://`*`exampleaccount`*`.cloudant.com`. Otherwise, your credentials are compromised.
- A user name: By default, the account name serves also as the administrator user name of the databases that are created in your account. Access is granted per database. By default, all databases that you create are only accessible to your account. If you are using a shared database that is accessible to multiple users, you need a user name that has sufficient permissions to write and read documents of the database.
- A password to access the account at `cloudant.com`.

For more information about creating an account at `cloudant.com`, see the sign up page for IBM Cloudant.

IBM MobileFirst Platform Cloudant Data Layer Local Edition

If you use a Cloudant Local installation instead of `cloudant.com`, you must configure the following information:

- The URL to access your Cloudant Local installation.

- A user name to access your Cloudant Local installation, which has sufficient permissions to write and read documents in the database.
- A password for this user in your Cloudant Local installation.

For more information about creating and administering users for Cloudant Local, see Configuring database-level security.

Permissions

The default permissions for a new account at `cloudant.com`, or at Cloudant Local, are sufficient for the MobileFirst Server administration.

The database can be created through the Cloudant dashboard. The user must have at least the following roles set for your database:

- `_reader`: Gives the user permission to read documents from the database.
- `_writer`: Gives the user permission to create and modify documents in the database.
- `_admin`: Gives the user permission to install and update design documents in the database.

  This permission is needed at least the first time for the application server to access the database after its creation. It is also needed the first time for the application server to access the database after a MobileFirst Server upgrade.

Unlike relational databases, there is no concept of tables in Cloudant, so there is no need to create any tables ahead of time or to set any permissions on tables.

If the database is not already created, it is automatically created when the MobileFirst Server administration is started on the application server. In this case, the user name must be set to the administrator user name of the Cloudant account, because only the administrator has permission to create new databases.

Default database name for MobileFirst Server administration

The default name of the administration database is `mfp_admin_db`. The database name can be changed through the JNDI property **`mfp.db.cloudant.adminDbName`**.

Configuration of the application server

IBM MobileFirst Platform Foundation for iOS 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

MobileFirst 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 page Configuring secure JMX connection to the Liberty profile.

- 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`.
- If you are going to use a Cloudant database for the administration or for a deployed project WAR file, configure the application server as described in "Configuring WebSphere Application Server Liberty profile for use with Cloudant."

What to do next

For more information about the optimization of MobileFirst Server, especially the tuning of the JVM memory allocation, see "Optimization and tuning of MobileFirst Server" on page 6-139.

Configuring WebSphere Application Server Liberty profile for use with Cloudant:

You must ensure that the truststore declared for the Liberty server through JVM properties recognizes the SSL certificate from the Cloudant server. The truststore declared in `server.xml` is irrelevant here.

Procedure

1. Determine the location of the truststore that is declared for the Liberty server through JVM properties:
   - Look into the file *WLP_DIR*/usr/servers/*servername*/jvm.options. If this file does not exist, look into the file *WLP_DIR*/etc/jvm.options. If you find an option of the form `-Djavax.net.ssl.trustStore=...`, the value is the truststore location that you are looking for.
   - If there is no option of the previously mentioned form, or if none of the files previously mentioned exist, you can find the truststore location in *JRE_DIR*/lib/security/cacerts.
   - If you are unsure about the truststore location, you can determine it as follows:
     a. Make sure that you have a file *WLP_DIR*/usr/servers/*servername*/jvm.options. If this file does not exist, create it as a copy of the file *WLP_DIR*/etc/jvm.options, or create an empty file, if neither of these two files exists.
     b. Add a line `-Djavax.net.debug=ssl` to this `jvm.options` file.
     c. Restart the Liberty server.
     d. In *WLP_DIR*/usr/servers/*servername*/logs/messages.log, look for lines that contain `0 trustStore is: ....`
     e. You can then remove the option `-Djavax.net.debug=ssl` that you added.
2. If the truststore location is a *JRE_DIR*/lib/security/cacerts file:
   - If the Cloudant server is on the internet, such as *account*.cloudant.com, or if it is a Cloudant Local server with a certificate that is signed by a trusted certificate authority, you have nothing to do, because the `cacerts` file already contains the root certificates for most internet servers.
   - Otherwise, do as follows:

a. Create a new truststore file that contains both the contents of the cacerts file and the signer certificate of the Cloudant server, by using the **keytool** command that is contained in the JRE.

b. Make sure that you have a file *WLP_DIR*/usr/servers/*servername*/jvm.options. If this file does not exist, create it as a copy of the file *WLP_DIR*/etc/jvm.options, or create an empty file, if neither of these two files exists.

c. Augment the file *WLP_DIR*/usr/servers/*servername*/jvm.options with options to set the JVM properties javax.net.ssl.trustStore, javax.net.ssl.trustStoreType, and javax.net.ssl.trustStorePassword, as documented in the JSSE Reference Guide, one per line.

3. If the truststore location is a different file, add the signer certificate of the Cloudant server to this truststore, by using the **keytool** command that is contained in the JRE.

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.
- The setenv.sh file is used if you start Apache Tomcat with <*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:

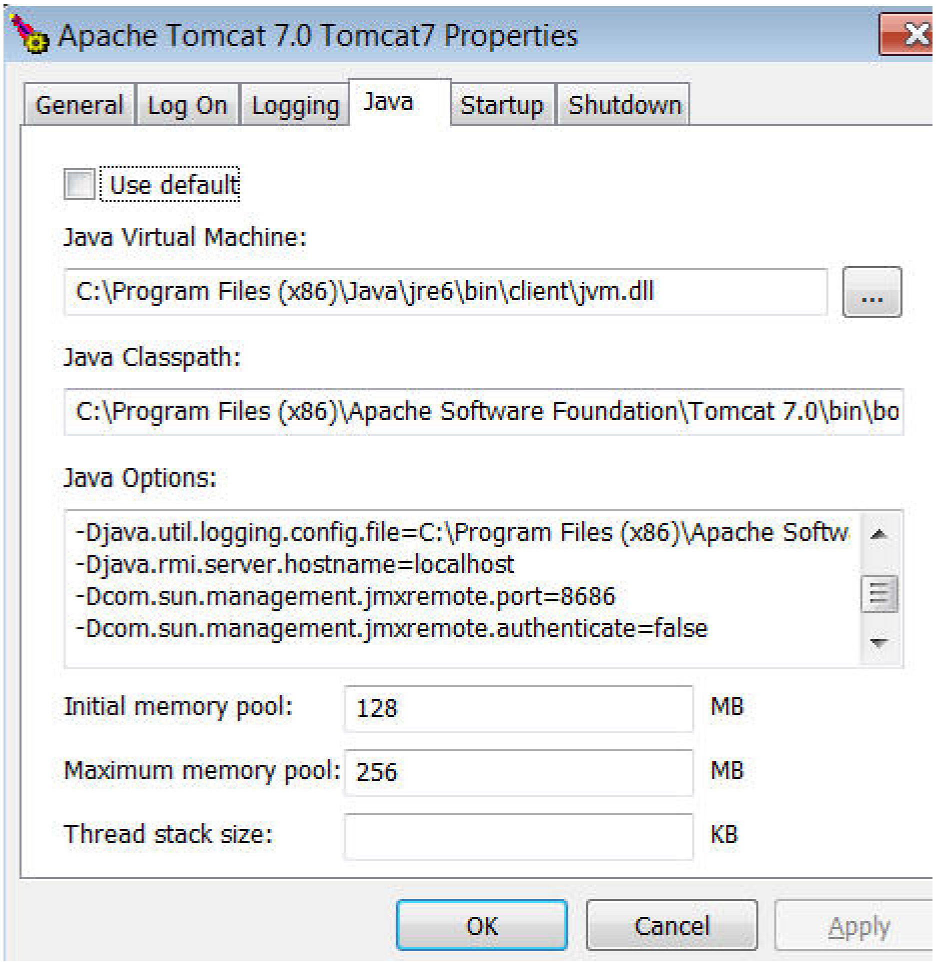
   ```
   <Context docBase="worklightadmin" path="/worklightadmin ">
       <Environment name="ibm.worklight.admin.rmi.registryPort" value="registryPort" type="java.l
       <Environment name="ibm.worklight.admin.rmi.serverPort" value="serverPort" type="java.lang.
   </Context>
   ```

   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.

5. If you are going to use a Cloudant database for the administration or for a deployed project WAR file, configure the application server as described in “Configuring Apache Tomcat for use with Cloudant” on page 6-56.

6. 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.



□ Use default

Java Virtual Machine:

C:\Program Files (x86)\Java\jre6\bin\client\jvm.dll

Java Classpath:

C:\Program Files (x86)\Apache Software Foundation\Tomcat 7.0\bin\bo

Java Options:

```
-Djava.util.logging.config.file=C:\Program Files (x86)\Apache Softw
-Djava.rmi.server.hostname=localhost
-Dcom.sun.management.jmxremote.port=8686
-Dcom.sun.management.jmxremote.authenticate=false
```

Initial memory pool: 128 MB

Maximum memory pool: 256 MB

Thread stack size: KB

What to do next

For more information about the optimization of MobileFirst Server, especially the tuning of the JVM memory allocation, see “Optimization and tuning of MobileFirst Server” on page 6-139.

Configuring Apache Tomcat for use with Cloudant:

You must ensure that the truststore declared for the Apache Tomcat server through JVM properties recognizes the SSL certificate from the Cloudant server.

Procedure

1. Determine the location of the truststore that is declared for the Apache Tomcat server through JVM properties:
   - If the file `TOMCAT_DIR/bin/setenv.sh` (on UNIX and Linux), or `TOMCAT_DIR/bin/setenv.bat` (on Windows) contains an augmentation of the `CATALINA_OPTS` variable with an option of the form `-Djavax.net.ssl.trustStore=...`, the value is the truststore location that you are looking for.
   - Otherwise, if you have a `CATALINA_OPTS` environment variable set, and it contains an option of the form `-Djavax.net.ssl.trustStore=...`, the value is the truststore location that you are looking for.

- If you cannot find the truststore location by any of the previously mentioned ways, then the truststore location is *JRE_DIR*/lib/security/cacerts

2. If the truststore location is a *JRE_DIR*/lib/security/cacerts file:
   - If the Cloudant server is on the internet, such as *account*.cloudant.com, or if it is a Cloudant Local server with a certificate that is signed by a trusted certificate authority, you have nothing to do, because the cacerts file already contains the root certificates for most internet servers.
   - Otherwise, do as follows:
     a. Create a new truststore file that contains both the contents of the cacerts file and the signer certificate of the Cloudant server, by using the **keytool** command that is contained in the JRE.
     b. Augment the file *TOMCAT_DIR*/bin/setenv.sh (on UNIX and Linux), or *TOMCAT_DIR*/bin/setenv.bat (on Windows), with options to set the JVM properties javax.net.ssl.trustStore, javax.net.ssl.trustStoreType, and javax.net.ssl.trustStorePassword, as documented in the JSSE Reference Guide.
3. If the truststore location is a different file, add the signer certificate of Cloudant server to this truststore, by using the **keytool** command that is contained in the JRE.

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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS uses the SOAP port by default. If both the SOAP and RMI ports are deactivated, IBM MobileFirst Platform Foundation for iOS does not run.
- RMI is only supported with a WebSphere Application Server Network Deployment. RMI is not supported with a stand-alone profile, or with a WebSphere Application Server server farm.
- You must activate Administrative and Application Security.
- If you are going to use a Cloudant database for the administration or for a deployed project WAR file, configure the application server as described in “Configuring WebSphere Application Server for use with Cloudant.”

What to do next

For more information about the optimization of MobileFirst Server, especially the tuning of the JVM memory allocation, see “Optimization and tuning of MobileFirst Server” on page 6-139.

Configuring WebSphere Application Server for use with Cloudant:

You must ensure that some SSL configuration in the WebSphere Application Server recognizes the SSL certificate from https://*username*.cloudant.com. To do so, you can add the signer of this certificate either to the default SSL configuration, or to an SSL configuration of its own. The second option provides for higher security. Similar instructions apply if you use a Cloudant Local server over HTTPS instead of `cloudant.com`.

To add this certificate to the default SSL configuration

1. Open the WebSphere Application Server console.
2. Go to **Security** > **SSL Certificates and Key Management**.
3. In **Related Items**, click **Key stores and certificates**
4. Select **NodeDefaultTrustStore**.
5. Select **Additional Properties** > **Signer certificates**.
6. Click **Retrieve from port**.
   a. Enter the Cloudant host name *`username`*`.cloudant.com`, where *`username`* is the user name of the Cloudant account, and the port, which is by default 443.

      Important: Do not enter the host name `cloudant.com`. This would not work, as it has a different signer certificate than *`username`*`.cloudant.com`.
   b. Select an alias, for example **`cloudant certificate signer`**.
   c. Click **Retrieve signer information**.
   d. Click **OK**.
7. Click **Save**.

To add the certificate signer to a new SSL configuration

1. First, create a truststore that contains only the signer of this certificate:
   a. Open the WebSphere Application Server console.
   b. Go to **Security** > **SSL Certificates and Key Management**.
   c. In **Related Items**, click **Key stores and certificates**.
   d. Set the **Keystore usages** combobox to **SSL keystores**.
   e. Click **New**, and enter the name `CloudantTruststore`, the path `cloudanttruststore` (this path is relative to `${CONFIG_ROOT}/etc`), and a password.
   f. Click **OK**.
   g. Click **Save**.
   h. Select the **CloudantTruststore** keystore.
   i. Select **Additional Properties** > **Signer certificates**.
   j. Click **Retrieve from port**.
      1) Enter the Cloudant host name *`username`*`.cloudant.com`, where *`username`* is the user name of the Cloudant account, and the port, which is by default 443.

         Important: Do not enter the host name `cloudant.com`. This would not work, as it has a different signer certificate than *`username`*`.cloudant.com`.
      2) Select an alias, for example **`cloudant certificate signer`**.
      3) Click **Retrieve signer information**.
      4) Click **OK**.
   k. Click **Save**.
2. If you are using WebSphere Application Server Network Deployment, copy the truststore file to all relevant nodes. In this case, the truststore file is `${CONFIG_ROOT}/etc/cloudanttruststore`.

3. Create an SSL configuration that uses this truststore:
   a. Open the WebSphere Application Server console.
   b. Go to **Security** > **SSL Certificates and Key Management**.
   c. In **Related Items**, click **SSL configurations**.
   d. Click **New**, and enter the name `CloudantConfig`, select as **Trust store name** the **CloudantTruststore**.
   e. Click **OK**.
   f. Click **Save**.
4. There are two ways of using this configuration:
   - You can define the JNDI property `mfp.db.cloudant.ssl.configuration` in the MobileFirst Server administration and project WAR applications, later.
   - You can create a dynamic outbound configuration:
     a. Open the WebSphere Application Server console.
     b. Go to **Security** > **SSL Certificates and Key Management**.
     c. In **Related Items**, click **Dynamic outbound endpoint SSL configurations**.
     d. Click **New**, and enter the name `MyCloudant` and a description.
     e. Enter a connection information: `HTTP`, `.cloudant.com`, `443`, and click **Add >>**.
     f. In the SSL configuration combination box, select **CloudantConfig**.
     g. Click **OK**.
     h. Click **Save**.

See also

- "Declaring the Cloudant SSL configuration for the MobileFirst Server administration" on page 6-100
- "Declaring the Cloudant SSL configuration for a project WAR file" on page 11-49

Configuring WebSphere Application Server V7.0:

If you use WebSphere Application Server V7.0, you must add a custom property to the web container settings.

Procedure

1. Click **Servers** > **Server Types** > **Application Servers**, and select the server for IBM MobileFirst Platform Foundation.
2. Click **Web Container Settings** > **Web container**.
3. Click **Custom properties**.
4. Click **New**.
5. Add the following property:
   - Name: `com.ibm.ws.webcontainer.invokerequestlistenerforfilter`
   - Value: `true`
6. Click **OK**.
7. Click **Save**.

Results

For more information, see the page PK91120: Servlet listeners not getting fired with servlet filters.

Installing MobileFirst Server administration with the Server Configuration Tool

You can use the Server Configuration Tool to install and configure MobileFirst 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 6-5.

About this task

Restriction:

- The Server Configuration Tool does not support server farms. Therefore, you cannot use this tool to install, upgrade, or configure a server farm.
- MobileFirst Server is not supported for production use on Mac OS X.

Procedure

1. Start the Server Configuration Tool.
   - On Linux: In the desktop menu, click **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Windows: In the **Start** menu, click **IBM MobileFirst Platform Server** > **Server Configuration Tool**.
   - On Mac OS X: In the **Finder**, double-click the file `*mf_server_install_dir*/shortcuts/configuration-tool.sh`.

     Note: The *mf_server_install_dir* placeholder represents the directory where you install MobileFirst Server. *mf_server* is the shortcut for MobileFirst Server.
2. Select **Create a MobileFirst Server Configuration**.
3. Name your configuration.
4. In the Configuration Description window:
   a. Enter the context root of the MobileFirst Administration REST service.

      The context root is used to create the URL to the MobileFirst 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 MobileFirst 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 MobileFirst Server administration component. If you install or upgrade the MobileFirst runtime environments with separate Ant files, this verification is particularly important because the **`environmentId`** attribute must match. For a server farm, all installations must also have the same **`environmentId`** attribute.

      The **`environmentId`** attribute is an attribute of the following Ant tasks:

- **installworklightadmin**, **updateworklightadmin**, and **uninstallworklightadmin**, which are documented at “Ant tasks for installation of MobileFirst Operations Console and Administration Services” on page 15-24.
- **configureapplicationserver**, **updateapplicationserver**, **unconfigureapplicationserver**, which are documented at “Ant tasks for installation of MobileFirst runtime environments” on page 15-32.

5. In the Console Settings window, enter the context root of the MobileFirst Operations Console.
6. In the Database Properties window:
   a. Select your database type: IBM DB2, MySQL, Oracle, or Cloudant.
   b. In the next window, enter the details to connect to the database instance, or to a valid Cloudant account.
   c. In the Database itional 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 6-47, enter the 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.

      If you selected a relational database, the Server Configuration Tool creates the database for you. If you select Cloudant as the database management system for your MobileFirst runtime environment, the Server Configuration Tool does not create a database.
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 MobileFirst Platform Foundation for iOS to that application server.
   c. Depending on your application server, proceed as follows:
      - 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 MobileFirst Operations Console
      - If the application server is WebSphere Application Server, select **Declare the WebSphere Administrator as an administrator of IBM MobileFirst Platform Operations Console** if you want to allow the WebSphere administrator to log in to the MobileFirst Operations Console.

      For more information about further configuration of security roles, see “Configuring user authentication for MobileFirst Server administration” on page 6-96.
8. When all the data is entered, click **Deploy**. The following effects take place:
   - For relational databases:
     – If the database administrator did not complete step “Optional creation of the administration database” on page 6-47, the database for the MobileFirst Server administration is created and access rights are granted to the user that is specified in the **database additional properties** window.

   - If the tables for MobileFirst administration do not exist in the database, they are created.
   - The MobileFirst administration components are installed in the application server and are connected to the database.
   - For Cloudant:
     - The application server is configured with the required information to connect to your Cloudant account. The database for the MobileFirst Server administration is created when the application server starts.
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 MobileFirst Operations Console, follow the steps in "Securing the MobileFirst Server administration" on page 6-151.
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 unsecured URL `<URL_TO_APPLICATION_SERVER_HTTP_PORT>/worklightconsole`.

What to do next

Install a MobileFirst runtime environment. For more information, see "Deploying, updating, undeploying, or upgrading MobileFirst Server by using the Server Configuration Tool" on page 11-11.

Using Ant tasks to install MobileFirst Server administration

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

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

If you did not manually create the database, you can use Ant tasks to create and configure your database for MobileFirst Server administration. If your database exists, you can perform only the configuration steps with Ant tasks.

Before you begin

Make sure that a database management system (DBMS) is installed and running on a database server, which can be the same computer, or a different computer.

Note: Apache Derby is not supported for production use.

If you want to start the Ant task from a computer on which MobileFirst Server is not installed, you must copy the following files to that computer:
- The library `*mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar` file.
- The Ant sample files that are in `*mf_server_install_dir*/WorklightServer/configuration-samples` folder.

Note: The *mf_server_install_dir* placeholder represents the directory where you installed MobileFirst Server.

About this task

Important: You use this procedure to create a relational database, if needed, and to populate it with the required tables. If you deploy with Cloudant as a database, you do not need to perform this procedure. However, you must ensure that you have a valid Cloudant account, as described in "Creating the Cloudant database for MobileFirst Server administration" on page 6-51. The MobileFirst administration Cloudant database is created, if needed, when the MobileFirst Server administration application is run for the first time.

If you did not create your database manually, as described in "Optional creation of the administration database" on page 6-47, complete steps 1 to 3.

If your relational database exists, you must create only the database tables. To create these tables, follow steps 4 to 7.

Procedure

If you did not create your database manually, complete steps 1 to 3.

1. Copy the Ant file that corresponds to your database. The files for creating a database are named after the following pattern:

   ```
   create-database-<dbms>.xml
   ```

   For more information, see table 1, Table 15-83 on page 15-67, in "Sample configuration files" on page 15-66.
2. Edit the Ant file and replace the placeholder values with the properties at the beginning of the file.
3. Run the following command to create the database.

   ```
   ant -f create-database-dbms.xml admdatabases
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.

If the database exists, then you must create only the database tables by completing the following steps.

4. Copy the sample Ant file that corresponds to both your application server and your DBMS. The files for configuring an existing database are named after this pattern:

   ```
   configure-<appServer>-<dbms>.xml
   ```

   For more information, see table 1, Table 15-83 on page 15-67, in "Sample configuration files" on page 15-66.
5. Edit the Ant file, and replace the placeholder values with the properties at the beginning of the file.
6. Run the following command to create the database.

   ```
   ant -f configure-<appServer>-<dbms>.xml admdatabases
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.
7. Save the Ant file. You need it to deploy the MobileFirst Operations Console and Administration Services with Ant tasks. For more information, see "Deploying the MobileFirst Operations Console and Administration Services with Ant tasks" on page 6-64. If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.

What to do next

Deploy MobileFirst Operations Console and the Administration Services, see "Deploying the MobileFirst Operations Console and Administration Services with Ant tasks."

See also:
- "Ant **configuredatabase** task reference" on page 15-14
- "Sample configuration files" on page 15-66

Deploying the MobileFirst Operations Console and Administration Services with Ant tasks:

Use Ant tasks to deploy the MobileFirst Operations Console and Administration Services to an application server, and configure data sources, properties, and database drivers that are used by IBM MobileFirst Platform Foundation for iOS.

Before you begin

1. Complete the procedure in "Creating and configuring the databases with Ant tasks" on page 11-15.
2. Run the Ant task on the computer where the application server is installed, or the Network Deployment Manager for WebSphere Application Server Network Deployment. If you want to start the Ant task from a computer on which MobileFirst Server is not installed, you must copy the following files to that computer:
   - The library `*mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar` file.
   - The Ant sample files that are in `*mf_server_install_dir*/WorklightServer/configuration-samples` folder.
3. If you use Cloudant via https as a database, and WebSphere Application Server full profile as an application server, declare the Cloudant signer certificate, as described in "Configuring WebSphere Application Server for use with Cloudant" on page 6-57.

Note: The *mf_server_install_dir* placeholder represents the directory where you installed MobileFirst Server.

Procedure

1. Copy the Ant file that corresponds to both your application server and your database management system (DBMS). The files for deploying a project WAR file are named after this pattern:

   ```
   configure-<appServer>-<dbms>.xml
   ```

   For more information, see table 1, Table 15-83 on page 15-67, in "Sample configuration files" on page 15-66.
2. Edit the Ant file, and replace the placeholder values with the properties at the beginning of the file.

   For WebSphere Application Server Liberty profile, the administration services require access to the RESTConnector, which is only accessible with Liberty administrator credentials. If you do not modify the `<jmx/>` tag, a new user named WorklightRESTUser is declared in the basic registry and is given administrator rights by declaring this user in the `<administrator-role/>` tag. You might have to modify the `<jmx/>` tag to define the Liberty administrator credentials for example if the Liberty administrator is identified using LDAP. In the Ant file, replace the empty <jmx/> tag by the following line.

```
<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 to the basic registry or to declare the user as a Liberty administrator.

3. To deploy the Administration Services and the MobileFirst Operations Console WAR files, run the following command:

   ```
   ant -f configure-<appServer>-<dbms>.xml adminstall
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts
4. Save the Ant file. You might need it later to apply a fix pack or perform an upgrade. For more information, see “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21. If you do not want to save the passwords, you can replace them by “************” (12 stars) for interactive prompting.

What to do next

Install a MobileFirst runtime environment. For more information, see “Using Ant tasks to deploy the project WAR file” on page 11-15.

See also:

- “Sample configuration files” on page 15-66
- “Ant tasks for installation of MobileFirst Operations Console and Administration Services” on page 15-24
- “Encrypting database password with Ant tasks for Liberty” on page 15-22

Manually installing MobileFirst Server administration

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

Configuring the DB2 database manually for the IBM MobileFirst Platform 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 MobileFirst Server administration” on page 6-48.
2. Create the tables in the database. This step is described in “Setting up your DB2 database manually for the MobileFirst Server administration” on page 6-66.
3. Perform the application server-specific setup as the following list shows.

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

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

About this task

Set up your DB2 database for the MobileFirst 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** You can change the name of the schema. This command can be run on an existing database whose page size is compatible with the one defined in step 3.
   ```
   db2 CONNECT TO WLADMIN
   db2 SET CURRENT SCHEMA = 'WLADMSC'
   db2 -vf product_install_dir/WorklightServer/databases/create-worklightadmin-db2.sql -t
   ```

Configuring Liberty profile for DB2 manually for MobileFirst Server administration:

You can set up and configure your DB2 database manually for MobileFirst Server administration with WebSphere Application Server Liberty profile.

Before you begin

Complete the DB2 Database Setup procedure before continuing.

Procedure

1. Add the DB2 JDBC driver JAR file to *$LIBERTY_HOME*/wlp/usr/shared/resources/db2.

   If that directory does not exist, create it. You can retrieve the file in one of two ways:
   - Download it from DB2 JDBC Driver Versions.
   - Fetch it from the *db2_install_dir*/java on the DB2 server directory.

2. Configure the data source in the *$LIBERTY_HOME*/wlp/usr/servers/worklightServer/server.xml file as follows:

   In this path, you can replace worklightServer by the name of your server.

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

   <!-- Declare the 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>
   ```

   The worklight value after **user=** is the name of the system user with CONNECT access to the **WLADMIN** database that you have previously created. The worklight value 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 computer).

   DB2 has a user name and password length limit of 8 characters for UNIX and Linux systems, and 30 characters for Windows.
3. You can encrypt the database password with the securityUtility program in <liberty_install_dir>/bin.

Configuring WebSphere Application Server for DB2 manually for MobileFirst Server administration:

You can set up and configure your DB2 database manually for the MobileFirst Server administration with WebSphere Application Server.

About this task

Complete the DB2 database setup procedure before continuing.

Note: The *was_install_dir* and *mf_server_install_dir* placeholders denote the directories where you installed WebSphere Application Server and MobileFirst Server.

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. 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 **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 MobileFirst 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 MobileFirst Server administration:

You can set up and configure your DB2 database manually for IBM MobileFirst Platform Server administration with the Apache Tomcat application server.

About this task

Before you continue, complete the DB2 database setup procedure.

Procedure

1. Add the DB2 JDBC driver JAR file.

   You can retrieve this JAR file in one of the following ways:

   - 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. Prepare an XML statement that defines the data source, as shown in the following code example.

   ```
   <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;"/>
   ```

   The **`worklight`** parameter after **`username=`** is the name of the system user with "CONNECT" access to the **`WLADMIN`** database that you previously created. The **`password`** parametere 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.

   DB2 enforces limits on the length of user names and passwords.

   - For UNIX and Linux systems: 8 characters
   - For Windows: 30 characters
3. Insert this statement in the `server.xml` file, as indicated in “Configuring Apache Tomcat for MobileFirst Server administration manually” on page 6-88.

Configuring the Apache Derby database manually for the IBM MobileFirst Platform 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 MobileFirst 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 MobileFirst Server administration"
   - "Configuring WebSphere Application Server for Derby manually for MobileFirst Server administration" on page 6-71
   - "Configuring Apache Tomcat for Derby manually for the MobileFirst Server administration" on page 6-73

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

You can set up your Apache Derby database for the MobileFirst Server administration manually.

About this task

Set up your `Apache Derby` database for the MobileFirst 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.

   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.

   For supported versions of Apache Derby, see "System requirements" on page 2-9.
   The script displays `ij` version number.
2. At the command prompt, enter the following commands:
   ```
   connect 'jdbc:derby:WLADMIN;user=WLADMIN;create=true';
   run 'product_install_dir/WorklightServer/databases/create-worklightadmin-derby.sql';
   quit;
   ```

Configuring Liberty profile for Derby manually for MobileFirst Server administration:

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

You can set up and configure your Apache Derby database manually for the MobileFirst Server administration with WebSphere Application Server.

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 `*product_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** data source 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 MobileFirst Server administration" on page 6-70.

   p. Click **OK**.

   q. Click **Save**.

   r. Click **administration 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 **administration Database** data source that you created.

   x. Optional: Only if you are not on the console of a WebSphere Application Server Deployment Manager, click **test connection**.

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 MobileFirst Server administration:

If you want to manually set up and configure your Apache Derby database for the IBM MobileFirst Platform 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 *product_install_dir*/WorklightServer/tools/lib/derby.jar to the directory $TOMCAT_HOME/lib.
2. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the server.xml file, as indicated in "Configuring Apache Tomcat for MobileFirst Server administration manually" on page 6-88

```
<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"/>
```

Configuring the MySQL database manually for the IBM MobileFirst Platform 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 MobileFirst Server administration" on page 6-49.
2. Create the tables in the database. This step is described in "Setting up your MySQL database manually for the MobileFirst Server administration."
3. Perform the application server-specific setup as the following list shows.

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

You can set up your MySQL database for the MobileFirst 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 product_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 MobileFirst Platform Foundation for iOS runs.

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

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

3. Add the following property to your MySQL option file: `innodb_log_file_size = 250M`

   For more information about **innodb_log_file_size**, see MySQL documentation: InnoDB Startup Options and System Variables.

Configuring Liberty profile for MySQL manually for MobileFirst Server administration:

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

Before you begin

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 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).

3. You can encrypt the database password with the securityUtility program in `<liberty_install_dir>/bin`.

Configuring WebSphere Application Server for MySQL manually for MobileFirst Server administration:

If you want to manually set up and configure your MySQL database for MobileFirst 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. 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 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 MobileFirst Server administration:

If you want to manually set up and configure your MySQL database for IBM MobileFirst Platform 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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the `server.xml` file, as indicated in "Configuring Apache Tomcat for MobileFirst Server administration manually" on page 6-88.

```
<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"/>
```

Configuring the Oracle database manually for the IBM MobileFirst Platform 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 MobileFirst Server administration" on page 6-50.
2. Create the tables in the database. This step is described in "Setting up your Oracle database manually for the MobileFirst Server administration."
3. Perform the application server-specific setup as the following list shows.

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

You can set up your Oracle database for the MobileFirst 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 computer, make sure that the database is started next time the Oracle installation is restarted. To this effect, make sure that 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 runtime database/schema, by using Oracle Database Control, proceed as follows:
     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: Unlocked
        Add system privilege: CREATE SESSION
        ```

```
Add system privilege: CREATE SEQUENCE
Add system privilege: CREATE TABLE
Add quota: Unlimited for tablespace USERS
```

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

  ```
  CONNECT SYSTEM/<SYSTEM_password>@ORCL
  CREATE USER WLADMIN IDENTIFIED BY password DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS;
  GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO WLADMIN;
  DISCONNECT;
  ```

3. Create the database tables for the runtime database and reports database:
   a. Using the Oracle SQLPlus command-line interpreter, create the tables for the IBM administration database by running the `create-worklightadmin-oracle.sql` file:

      ```
      CONNECT WLADMIN/WLADMIN_password@ORCL
      @product_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 MobileFirst Server administration:

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

Before you begin

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 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).

Note: For more information on how to connect the Liberty server to the Oracle database with a service name, or with a URL, see the WebSphere Application Server Liberty Core 8.5.5 documentation, section **properties.oracle**.

3. You can encrypt the database password with the securityUtility program in <liberty_install_dir>/bin.

Configuring WebSphere Application Server for Oracle manually for the MobileFirst Server administration:

If you want to manually set up and configure your Oracle database for the MobileFirst 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 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 6-20. 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 MobileFirst Server administration:

If you want to manually set up and configure your Oracle database for IBM MobileFirst Platform 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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the `server.xml` file, as indicated in "Configuring Apache Tomcat for MobileFirst Server administration manually" on page 6-88

   ```
   <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"/>
   ```

   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.

Configuring the Cloudant database manually for MobileFirst Server administration:

You configure the Cloudant database by setting JNDI properties in the appropriate application server.

There is no manual configuration required for a Cloudant database. If the database for the MobileFirst Server administration does not exist, and if the Cloudant user that is provided to the MobileFirst Server administration is the Cloudant administrator, then the database is automatically created when the MobileFirst Server administration is started. If the user that is provided to the MobileFirst Server administration is not allowed to create a database, you must create a database with the names that are documented at "Creating the Cloudant database for MobileFirst Server administration" on page 6-51.

For more information about the MobileFirst Server administration configuration with Cloudant, see "Deploying the Administration Services and MobileFirst Operations Console and configuring the application server manually"

Deploying the Administration Services and MobileFirst Operations Console and configuring the application server manually:

The procedure to deploy the Administration services and IBM MobileFirst Platform Operations Console manually to an application server depends on your application server.

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

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

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

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

About this task

In addition to modifications for the databases, which are described in "Manually installing MobileFirst Server administration" on page 6-65, you must make the following modifications to the `server.xml` file.

Note: In the following procedure, when the example uses the `worklight.war` file name, use the name of your MobileFirst project, for example, `myProject.war`.

As the `server.xml` file contains passwords, you need to protect this file. If you must prevent other users on the same computer from accessing these passwords, 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 server.xml`
- On Windows: `cacls server.xml /P Administrators:F %USERDOMAIN%\%USERNAME %:F`

You can also encode the passwords with the Liberty profile `securityUtility` program, which is provided in the *`liberty_install_dir`*`/bin` directory.

Procedure

1. Ensure that the `<featureManager>` element contains at least the following `<feature>` elements:
   - For WebSphere Application Server Liberty profile V8.5.0.x:
     ```
     <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>
     ```
   - For WebSphere Application Server Liberty profile V8.5.5.0 and later:
     ```
     <feature>jdbc-4.0</feature>
     <feature>appSecurity-2.0</feature>
     <feature>restConnector-1.0</feature>
     ```
2. Follow the instructions from the IBM WebSphere Application Server Liberty Core 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.host`** is the host name for the JMX REST connection.
   - **`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 on page 6-82.
- **ibm.worklight.admin.jmx.pwd** is the password of that user. This password can be encoded with the Liberty profile securityUtility program. The supported encoding types are xor and aes (only with the default key).

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

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

5. Declare a thread pool: Add the following <executor> declaration, or if the server.xml file has an <executor> declaration already, modify its coreThreads and maxThreads values accordingly.

   ```
   <!-- Thread pool -->
   <executor name="LargeThreadPool" id="default"
             coreThreads="200" maxThreads="400" keepAlive="60s"
             stealPolicy="STRICT" rejectedWorkPolicy="CALLER_RUNS"/>
   ```

6. Copy the following WAR files to the apps directory of the Liberty server: *product_install_dir*/WorklightServer/worklightadmin.war and *product_install_dir*/WorklightServer/worklightconsole.war.

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

7. Declare the Administration Services and MobileFirst Operations 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>
         <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.jar"/>
       </commonLibrary>
     </classloader>
   </application>

   <!-- Declare the MobileFirst Operations
   Console application. -->
   <application id="worklightconsole" name="worklightconsole" location="worklightconsole.war" typ
     <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>
   </application>

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

   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 MobileFirst Operations Console application, in this example `worklightconsole`. The value of this property is the end point to the MobileFirst administration.

The syntax `"*://*:*/worklightadmin"` means that the URL is the same as the one that is used to contact the MobileFirst Operations Console. However, the context root of the MobileFirst Operations Console is replaced by `worklightadmin`.

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

8. 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. This ID is declared in the procedure that is documented in “Configuring Liberty profile for Oracle manually for MobileFirst Server administration” on page 6-78.

9. If the database is Cloudant, add the following JNDI properties:

```
<!-- Declare the JNDI properties for the  Administration Services application  for Cloudant. -->
<jndiEntry jndiName="worklightadmin/mfp.db.cloudant.url" value='"https://wluser.cloudant.com:443"
<jndiEntry jndiName="worklightadmin/mfp.db.cloudant.username" value='"wluser"'/>
<jndiEntry jndiName="worklightadmin/mfp.db.cloudant.password" value='"password"'/>
```

   a. The JNDI properties are prefixed by the context root of the Administration Services application, in this example `worklightadmin`. If you select another value in a previous step, replace `worklightadmin` with that value.
   b. Replace `https://`*`wluser`*`.cloudant.com:443` with the URL to connect to Cloudant.
   c. Replace *`wluser`* with the name of the user who accesses the MobileFirst Server administration database. If this user is not allowed to create a database, you must create a database manually, with the names that are documented at “Creating the Cloudant database for MobileFirst Server administration” on page 6-51.
   d. Replace *`password`* with the password that is associated with this user. This password can be encoded with the Liberty profile `securityUtility` program. The supported encoding types are `xor` and `aes` (only with the default key).

   Note: The default database name is `mfp_admin_db`. If you want to change the database name, add the following JNDI property:

```
<jndiEntry jndiName="worklightadmin/mfp.db.cloudant.adminDbName" value='"your_db_name"'/>
```

   For a complete list of the JNDI properties that you can set, see “List of JNDI properties for MobileFirst Server administration” on page 6-101.

Configuring WebSphere Application Server for MobileFirst Server administration manually:

To configure WebSphere Application Server for IBM MobileFirst Platform Server administration manually, you must configure variables, custom properties, and class loader policies.

Before you begin

These instructions assume that a stand-alone profile exists with an application server named “Worklight” and that the server is using the default ports.

If you use Cloudant through https as a database, and WebSphere Application Server full profile as an application server, declare the Cloudant signer certificate, as described in “Configuring WebSphere Application Server for use with Cloudant” on page 6-57.

Procedure

1. Log on to the WebSphere Application Server administration console for your MobileFirst 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 IBM MobileFirst Platform Foundation for iOS.
   - 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 MobileFirst applications to **parent-first**.
4. For Derby, DB2, MySQL, or Oracle, create the MobileFirst Server administration JDBC data source and provider. See the instructions in the appropriate subsection in “Manually installing MobileFirst Server administration” on page 6-65.

   Note: This step is not relevant for Cloudant.
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 MobileFirst Server installation directory `product_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 on page 6-85. If you want to use Cloudant, select the default data source. The data source is not used in this case.

   To enter a default data source, enter `DefaultDatasource` in the **Name** field of the Target Resource JNDI. If the `DefaultDatasource` does not exist in your application server, a message displays the following message when you quit the page: "**Application Resource Warning**". Ignore the warning and click **Continue**.

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. If you want to use Cloudant as the database for the MobileFirst Server administration, set the variables as follows:

   - `mfp.db.cloudant.url`: the URL to connect to Cloudant
   - `mfp.db.cloudant.username`: the Cloudant account user name

     Note: If this user is not allowed to create a database, you must create a database manually, with the names that are documented at "Creating the Cloudant database for MobileFirst Server administration" on page 6-51.
   - `mfp.db.cloudant.password`: the Cloudant account password
   - The default database name is `mfp_admin_db`. If you want to change the database name, set the following JNDI property: `mfp.db.cloudant.adminDbName`
   - For a complete list of the JNDI properties that you can set, see "List of JNDI properties for MobileFirst Server administration" on page 6-101.

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

m. 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 IBM MobileFirst Platform Operations 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 MobileFirst Server installation directory `*product_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 MobileFirst Operations 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**.

What to do next

You must configure user authentication, as described in "Configuring user authentication for MobileFirst Server administration" on page 6-96. Then, you can access the MobileFirst 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`).

For more steps to configure MobileFirst Server administration, see "Configuring WebSphere Application Server full profile for MobileFirst Server administration" on page 6-98.

Configuring Apache Tomcat for MobileFirst Server administration manually:

To configure Apache Tomcat for the MobileFirst 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

Prerequisites:

- Configure the database for MobileFirst Server administration. For more information about various databases configuration, see "Manually installing MobileFirst Server administration" on page 6-65.
- Define the `CATALINA_OPTS` options to enable Java Management Extensions (JMX) as described in "Configuring Apache Tomcat" on page 6-54.

Procedure

1. Edit *`tomcat_install_dir`*`/conf/server.xml` file.

   a. Uncomment the following element, which is initially commented out: `<Valve className="org.apache.catalina.authenticator.SingleSignOn" />`.

   b. Declare the MobileFirst Operations 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.S

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

```
</Context>

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

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

<!-- Declare the user registry for the MobileFirst 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, for Derby, DB2, MySQL, and Oracle, you must uncomment and complete the `<Resource>` element to declare the administration database as described in one of the following sections:

- "Configuring Apache Tomcat for DB2 manually for MobileFirst Server administration" on page 6-69
- "Configuring Apache Tomcat for Derby manually for the MobileFirst Server administration" on page 6-73
- "Configuring Apache Tomcat for MySQL manually for MobileFirst Server administration" on page 6-76
- "Configuring Apache Tomcat for Oracle manually for MobileFirst Server administration" on page 6-80

c. If you want to use Cloudant as the database for the MobileFirst Server administration, add the following JNDI properties inside the `<Context docBase="worklightadmin" path="/worklightadmin">` element:

```
<Environment name="mfp.db.cloudant.url" value="https://wluser.cloudant.com:443" type="java.
<Environment name="mfp.db.cloudant.username" value="wluser" type="java.lang.String" overrid
<Environment name="mfp.db.cloudant.password" value="password" type="java.lang.String" overr
```

- Replace `https://wluser.cloudant.com:443` with the URL to connect to Cloudant.
- Replace *wluser* with the name of the user who accesses the MobileFirst Server administration database.

  Note: If this user is not allowed to create a database, you must create a database manually, with the names that are documented at "Creating the Cloudant database for MobileFirst Server administration" on page 6-51.
- Replace *password* with the password that is associated with this user.

Note: The default database name is `mfp_admin_db`. If you want to change the database name, add a JNDI property:

```
<Environment name="mfp.db.cloudant.adminDbName" value="your_db_name" type="java.lang.String
```

For a complete list of the JNDI properties that you can set, see "List of JNDI properties for MobileFirst Server administration" on page 6-101.

2. Copy the MobileFirst Server administration WAR files to Tomcat.
   - On UNIX and Linux systems:

     ```
     cp product_install_dir/WorklightServer/*.war tomcat_install_dir/webapps
     ```
   - On Windows systems:

     ```
     copy /B product_install_dir\WorklightServer\worklightconsole.war tomcat_install_dir\webapps\
     copy /B product_install_dir\WorklightServer\worklightadmin.war tomcat_install_dir\webapps\wo
     ```
3. Start Tomcat.

What to do next

For more steps to configure the MobileFirst Server administration, see "Configuring Apache Tomcat for MobileFirst Server administration" on page 6-100.

Installations that are not stand alone: information for experts:

You can configure a clustered installation or a server-farm installation.

The other topics in "Deploying the Administration Services and MobileFirst Operations Console and configuring the application server manually" on page 6-81 explain how to configure MobileFirst Server administration for stand-alone web servers. Other options are a clustered installation or a server farm installation. In these installations, MobileFirst Server administration runs on several servers with a load balancer in front.

For installation of server farms, see "Installing a server farm" on page 6-127.

Clustered installations are currently only supported for WebSphere Application Server full profile.

Cluster mode

The JNDI property **`ibm.worklight.topology.clusterMode`** specifies the type of installation. You can set it to:

- `Standalone`
- `Cluster`
- `Farm`

If cluster mode is not set, the system tries to determine the cluster mode automatically, but the heuristic might fail. Therefore, you should specify this property correctly.

The lock master for the Cloudant database

In a server farm or a clustered installation, several servers run MobileFirst Server administration in parallel. For some operations, synchronization, among all servers in the farm or among all nodes in a cluster, is required. For example, when an application or adapter is deployed through MobileFirst Operations Console on one server, the other servers are not allowed to have the same operation done at the same time.

Synchronization is implemented through a lock mechanism similar to Java locks. Cloudant does not support database locks, so the chosen lock mechanism depends on the database and the cluster mode.

- • If the database is Derby, DB2, MySQL, or Oracle, a database lock mechanism is used. This mechanism does not require extra configuration.
- • If the database is Cloudant and the cluster mode is `Standalone`, an in-memory lock mechanism is used. This mechanism does not require extra configuration.
- • If the database is Cloudant and the cluster mode is `Cluster` or `Farm`, a distributed lock mechanism is used that requires one of the servers to be the lock master.

When the database is Cloudant and the lock master is required, the system tries to detect the lock master automatically. In a farm, if the server that acts as current lock master is shut down, then another server in the farm automatically takes the role of lock master.

Optionally, you can specify the lock master through the property **ibm.worklight.admin.lock.master** with the possible value true or false. In this case, it must be specified with true for only one of the servers and with false for all the other servers. The property can be specified as a JNDI property for the servers of a farm or as a Java Virtual Machine (JVM) property in a WebSphere Application Server full profile cluster, because, in such a cluster, all cluster members have the same JNDI properties.

If the property **ibm.worklight.admin.lock.master** is specified, note that:

- When the farm or cluster is started, the other servers wait until the lock master is started. In a server farm, you should start the lock master first.
- The lock master must be running to permit any deployment operations in MobileFirst Operations Console.
- The other servers do not automatically take the role of lock master when the lock master is shut down.

Defining the endpoint of the MobileFirst Administration services

If circumstances require the parameters of the endpoint definition to be changed, you must configure properties of the web application server appropriately.

MobileFirst Operations Console must be able to locate the MobileFirst 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, host name, and port are the same as those defined in the web application server used to access MobileFirst Operations Console; the context root of the MobileFirst Administration REST services is worklightadmin. When the context root of the MobileFirst 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 MobileFirst Operations Console, the externally accessible endpoint (protocol, host name, 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 MobileFirst Operations Console is accessed with the external address (wrklght.net).

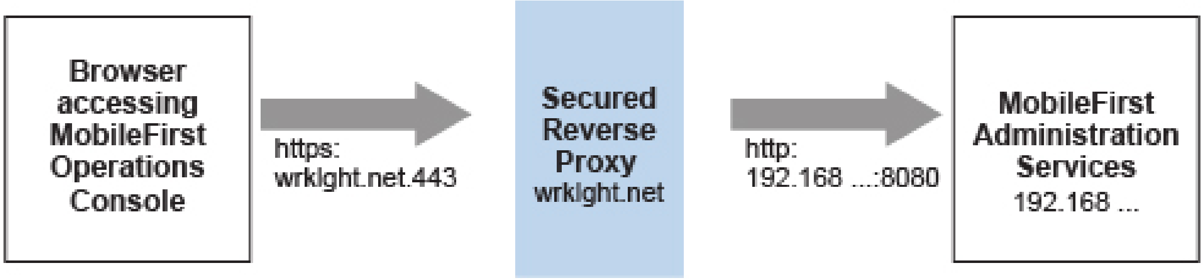

Browser accessing MobileFirst Operations Console
https: wrklght.net.443
Secured Reverse Proxy wrklght.net
http: 192.168 ...:8080
MobileFirst Administration Services 192.168 ...

Figure 6-7. Configuration with secured reverse proxy

Table 6-21. The endpoint properties

Property name	Purpose	Example
ibm.worklight.admin.endpoint	This property enables MobileFirst Operations Console to locate the MobileFirst 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 MobileFirst Administration services use the same value as MobileFirst Operations Console. For example: *://*:*/wladmin means use the same protocol, host, and port as MobileFirst Operations Console, but use wladmin as context root. This property must be specified for the MobileFirst Operations 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 MobileFirst Administration services. This property must be specified for the MobileFirst Administration services application.	https

Table 6-21. The endpoint properties (continued)

Property name	Purpose	Example
ibm.worklight.admin.proxy.host	If external access is required, this property specifies the host name for external browsers to access the MobileFirst Administration services. This property must be specified for the MobileFirst Administration services application.	wrklght.net
ibm.worklight.admin.proxy.port	If external access is required, this property specifies the port for external browsers to access the MobileFirst Administration services. This property must be specified for the MobileFirst Administration services application.	443

Configuring the endpoint (WebSphere Application Server full profile):

Configure the endpoint of the application resources in the environment entries of MobileFirst Operations Console and the MobileFirst Administration services application.

About this task

Follow this procedure when you must change the endpoint of the MobileFirst 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 host name.
   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 MobileFirst Administration services; That is, the URI of the `worklightadmin.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 MobileFirst Administration services use the same value as MobileFirst Operations Console. For example, *://*:*/wladmin means use the same protocol, host, and port as MobileFirst Operations 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 MobileFirst Server administration” on page 6-101.

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 MobileFirst 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 MobileFirst Server administration” on page 6-101.

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.

```
<jndiEntry jndiName="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 MobileFirst Operations 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 MobileFirst Server are running in the same web application server, this technique is particularly useful. If you have only one instance of MobileFirst 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 MobileFirst Administration services use the same value as MobileFirst Operations Console. For example, `*://*:*/wladmin` means use the same protocol, host, and port as MobileFirst Operations 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 MobileFirst 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 MobileFirst Server administration" on page 6-101.

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 MobileFirst Operations Console application:

```
<Environment name="ibm.worklight.admin.endpoint" value="https://wrklght.net:443/worklightadmin"
    type="java.lang.String" override="false"/></p>
```

For **ibm.worklight.admin.endpoint**, you can use the asterisk (*) character as wildcard for specifying that the MobileFirst Administration services use the same value as MobileFirst Operations Console. For example, `*://*:*/wladmin` means use the same protocol, host, and port as MobileFirst Operations Console, but use `wladmin` as context root.

In the `<context>` section of the MobileFirst 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="false"/>
```

Configuring user authentication for MobileFirst Server administration

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

The MobileFirst Server administration requires user authentication.

You configure user authentication after the installer deploys the MobileFirst Server administration web applications in the web application server.

The MobileFirst Server administration has the following Java Platform, Enterprise Edition (Java EE) security roles defined:

worklightadmin
worklightdeployer
worklightoperator
worklightmonitor

You must map the roles to the corresponding sets of users. The **worklightmonitor** role can view data but cannot change any data. The purpose of the roles is illustrated by the following table.

Table 6-22. MobileFirst Roles and Functionality - Production Server

	Administrator	Deployer	Operator	Monitor
Java EE security role.	**worklightadmin**	**worklightdeployer**	**worklightoperator**	**worklightmonitor**
Deployment				
Deploy an application.	Yes	Yes	No	No
Deploy an adapter.	Yes	Yes	No	No
MobileFirst Server Management				
Configure runtime settings.	Yes	Yes	No	No
Application Management				
Upload new MobileFirst application.	Yes	Yes	No	No

Table 6-22. MobileFirst Roles and Functionality - Production Server (continued)

	Administrator	**Deployer**	**Operator**	**Monitor**
Remove MobileFirst application.	Yes	Yes	No	No
Upload new MobileFirst adapter.	Yes	Yes	No	No
Remove MobileFirst adapter.	Yes	Yes	No	No
Turn on or off application authenticity testing for an application.	Yes	Yes	No	No
Change properties on MobileFirst application status: Active, Active Notifying, and Disabled.	Yes	Yes	Yes	No
Lock an application so the new artifacts cannot be used for a version.	Yes	Yes	Yes	No
Notifications				
Unsubscribe a device from SMS notification.	Yes	Yes	Yes	Yes
Configure Push.	Yes	Yes	Yes	Yes
Logging				
Enable and disable device logging remotely.	Yes	Yes	Yes	No
Configure log levels.	Yes	Yes	Yes	No
Disable the specific device, marking the state as lost or stolen so that access from any of the applications on that device is blocked.	Yes	Yes	Yes	No

Table 6-22. MobileFirst Roles and Functionality - Production Server (continued)

	Administrator	Deployer	Operator	Monitor
Disable a specific application, marking the state as disabled so that access from the specific application on that device is blocked.	Yes	Yes	No	No

If you choose to use an authentication method through a user repository such as LDAP, you can configure the MobileFirst Server administration so that you can use users and groups with the user repository to define the Access Control List (ACL) of the MobileFirst 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 MobileFirst Server administration:

Configure security by mapping the MobileFirst Server administration Java EE 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 MobileFirst Server administration:

Configure the Java EE security roles of the MobileFirst 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 MobileFirst Server administration:

You must configure the Java EE security roles for the MobileFirst Server administration on the Apache Tomcat web application server.

Procedure

1. If you installed the MobileFirst 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.

Declaring the Cloudant SSL configuration for the MobileFirst Server administration

If you install the MobileFirst Server administration into WebSphere Application Server or WebSphere Application Server Network Deployment, you use a Cloudant database, and meet specific criteria described in this topic, you must declare the Cloudant SSL configuration.

About this task

You must perform this procedure if you meet all of the following criteria:

- You are installing the MobileFirst Server administration into WebSphere Application Server or WebSphere Application Server Network Deployment.
- You use a Cloudant database for the MobileFirst Server administration.
- The Cloudant database server URL uses the https protocol.
- You created a separate SSL configuration for Cloudant, as described in “Configuring WebSphere Application Server for use with Cloudant” on page 6-57, but did not create a dynamic outbound configuration for it.

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. Set the value of the property mfp.db.cloudant.ssl.configuration to the name of the SSL configuration that you created.
6. Click **OK**.
7. Click **Save**.

List of JNDI properties for MobileFirst Server administration

When you configure MobileFirst Server Administration Services and MobileFirst Operations Console for your application server, you set optional or mandatory JNDI properties, in particular for Java Management Extensions (JMX).

JNDI properties for MobileFirst Administration Services

The following properties can be set on the Administration Services web application `worklightadmin.war`.

Table 6-23. JNDI properties for Administration Services: JMX

Property	Optional or 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.	WebSphere Application Server Liberty profile: User name for the JMX REST connection. WebSphere Application Server Farm: User name for the SOAP connection. WebSphere Application Server Network Deployment: user name of the WebSphere administrator if the virtual host mapped to the MobileFirst server administration application is not the default host.

Table 6-23. JNDI properties for Administration Services: JMX (continued)

Property	Optional or mandatory	Description	Restrictions
ibm.worklight.admin.jmx.pwd	Optional	User password for the JMX REST connection.	WebSphere Application Server Liberty profile: User password for the JMX REST connection. WebSphere Application Server Farm: User password for the SOAP connection. WebSphere Application Server Network Deployment: User password of the WebSphere administrator if the virtual host that is mapped to the MobileFirst Server server administration application is not the default host.
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 6-24. JNDI properties for Administration Services: time out

Property	Optional or 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 6-24. JNDI properties for Administration Services: time out (continued)

Property	Optional or mandatory	Description
ibm.worklight.admin.actions.commitRejectTimeout	Optional	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 6-25. JNDI properties for Administration Services: logging

Property	Optional or 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 6-26. JNDI properties for Administration Services: proxies

Property	Optional or mandatory	Description
ibm.worklight.admin.proxy.port	Optional	If the MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst Administration server. Typically, this property is the address of the proxy.

Table 6-27. JNDI properties for Administration Services: topologies

Property	Optional or mandatory	Description
ibm.worklight.admin.audit	Optional.	Set this property to false to disable the audit feature of the MobileFirst Operations 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 MobileFirst 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 6-27. JNDI properties for Administration Services: topologies (continued)

Property	Optional or mandatory	Description
ibm.worklight.admin.hsts	Optional.	Set to true to enable HTTP Strict Transport Security according to RFC 6797.
ibm.worklight.topology.platform	Mandatory	Server type. Valid values: • Liberty • WAS for WebSphere Application Server • Tomcat If you do not set the value, the application tries to guess the server type.
ibm.worklight.topology.clustermode	Mandatory	In addition to the server type, specify here the server topology. Valid values: • Standalone • Cluster • Farm The default value is Standalone.

Table 6-27. JNDI properties for Administration Services: topologies (continued)

Property	Optional or mandatory	Description
ibm.worklight.admin.lock.master	Optional	In cluster and farm topologies, this property determines which server is the lock master when the Cloudant database is used. For synchronization, the system requires a locking mechanism that works across all servers in the farm or cluster. SQL data bases have native locking facilities, but Cloudant has none. Therefore, with Cloudant only one of the servers in the farm or cluster provides the locking mechanism. This server is called the lock master. Alternative ways of using this property are available: • You set this property to `auto` on all servers. The locking master is chosen and updated dynamically, depending on server availability. • You set this property to `true` on one server only and to `false` on all the other servers. In this configuration, the server where the property has the value `true` is the lock master. In WebSphere Application Server Network Deployment topologies, setting the property to true must be done through a JVM property of the server. **Note:** This use creates a single point of failure. When the lock master is unavailable, MobileFirst Administration does not function. The default value is `auto`.

Table 6-27. JNDI properties for Administration Services: topologies (continued)

Property	Optional or mandatory	Description
ibm.worklight.admin.lock.master.detection.delay	Optional	This property determines the time in milliseconds to wait on startup until Cloudant is ready, before any locking operation can take place. This property is only needed in cluster or farm topology when the database is Cloudant. For synchronization, the system requires a locking mechanism that works across all servers in the farm or cluster. SQL databases have native locking facilities, but Cloudant has none. Therefore, with Cloudant one of the servers in the farm or cluster provides the locking mechanism. This lock master server can be selected automatically with the help of the database. This automatic selection requires a small delay, similar to the setting of **mfp.db.cloudant.afterWrite.delay**, to ensure that the Cloudant database is in a consistent state. This delay occurs only once when the server starts. Reasonable values are between 1 and 10 seconds. Negative values are ignored. The default value is 3000 (three seconds).
ibm.worklight.admin.lock.master.detection.timeout	Optional	This property determines the timeout in seconds for the detection of the lock master. This property is only needed in cluster or farm topology when the database is Cloudant. For synchronization, the system requires a locking mechanism that works across all servers in the farm or cluster. SQL databases have native locking facilities, but Cloudant has none. Therefore, with Cloudant one of the servers in the farm or cluster provides the locking mechanism. During startup, the lock master server must come alive before all the other servers. Therefore, the other servers wait for this server before they complete their startup. This timeout value specifies the maximum time another server waits for the lock master to come alive. Negative values are ignored. The default value is 120 (two minutes).

Table 6-27. JNDI properties for Administration Services: topologies (continued)

Property	Optional or mandatory	Description
`ibm.worklight.admin.lock.master.connect.timeout`	Optional	This property determines the timeout in seconds for the connection to the lock master. This property is only needed in cluster or farm topology when the database is Cloudant. For synchronization, the system requires a locking mechanism that works across all servers in the farm or cluster. SQL databases have native locking facilities, but Cloudant has none. Therefore, with Cloudant one of the servers in the farm or cluster provides the locking mechanism. That server is called the lock master. The timeout value specifies the maximum time another server waits for the lock master to respond to requests to take a lock. Negative values are ignored. The default value is 10 (seconds).
`ibm.worklight.admin.farm.heartbeat`	Optional	This property enables you to set in minutes the heartbeat rate that is used in server farm topologies. The default value is 2 minutes. In a server farm, all members must use the same heartbeat rate. If you set or change this JNDI value on one server in the farm, you must also set the same value on every other server in the farm. For more information, see "Lifecycle of a server farm node" on page 6-134.
`ibm.worklight.admin.farm.missed.heartbeats.timeout`	Optional	This property enables you to set the number of missed heartbeats of a farm member before the status of the farm member is considered to be failed or down. The default value is 2. In a server farm all members must use the same missed heartbeat value. If you set or change this JNDI value on one server in the farm, you must also set the same value on every other server in the farm. For more information, see "Lifecycle of a server farm node" on page 6-134.
`ibm.worklight.admin.farm.reinitialize`	Optional	A Boolean value (true or false) for re-registering or re-initializing the farm member.

Table 6-27. JNDI properties for Administration Services: topologies (continued)

Property	Optional or mandatory	Description
ibm.worklight.admin.cloudant.dashboard.url	Optional	This property defines the URL of the Cloudant dashboard, such as the dashboard of the Cloudant account that you use for the MobileFirst data proxy. If this property is set, a link will be displayed in the header of MobileFirst Operations Console.

Table 6-28. JNDI properties for Administration Services: relational database

Property	Optional or mandatory	Description
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.openjpa.ConnectionDriverName	Optional 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	Optional 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	Optional 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	Optional 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.

Table 6-28. JNDI properties for Administration Services: relational database (continued)

Property	Optional or mandatory	Description
`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.admin.db.type`	Optional	This property defines the type of database. The default value is inferred from the connection URL.

Table 6-29. JNDI properties for Administration Services: IBM Cloudant database.

Property	Optional or mandatory	Description
`mfp.db.cloudant.url`	Optional	This property defines the URL of the Cloudant account used to store the database. The default value is https://*username*.cloudant.com.
`mfp.db.cloudant.username`	Optional Conditionally mandatory	This property defines the user name of the Cloudant account used to store the database. If this property is not defined, a relational database is used.
`mfp.db.cloudant.password`	Optional Conditionally mandatory	This property defines the password of the Cloudant account used to store the database. This property must be set when **`mfp.db.cloudant.username`** is set.
`mfp.db.cloudant.ssl.authentication`	Optional	This property specifies whether the SSL certificate chain validation and host name verification are enabled for HTTPS connections to the Cloudant database. The value is a Boolean value (true or false). The default value is `true`. **Note:** Setting this property to `false` creates security risks.

Table 6-29. JNDI properties for Administration Services: IBM Cloudant database (continued).

Property	Optional or mandatory	Description
mfp.db.cloudant.ssl.configuration	Optional	This property applies to WebSphere Application Server full profile only. For HTTPS connections to the Cloudant database, it specifies the name of an SSL configuration in the WebSphere Application Server configuration to use when no configuration is specified for the host and port.
mfp.db.cloudant.proxyHost	Optional	This property defines the host name of an HTTP proxy for the connection to the Cloudant database server.
mfp.db.cloudant.proxyPort	Optional	This property defines the port of an HTTP proxy for the connection to the Cloudant database server.
mfp.db.cloudant.adminDbName	Optional	This property defines the name of the database for MobileFirst Administration Services in the Cloudant account. The name must start with a lowercase letter and contain only lowercase letters and any of the following characters: 0-9 , $ - _ The default name is `mfp_admin_db`.
mfp.db.cloudant.connectTimeout	Optional	This property defines the timeout in milliseconds for establishing a network connection for Cloudant. A value of zero means an infinite timeout. A negative value means the default value (no override).
mfp.db.cloudant.socketTimeout	Optional	This property defines the timeout in milliseconds for detecting the loss of a network connection for Cloudant. A value of zero means an infinite timeout. A negative value means the default value (no override).
mfp.db.cloudant.maxConnections	Optional	This property defines the maximum number of simultaneous connections to the Cloudant database.

Table 6-29. JNDI properties for Administration Services: IBM Cloudant database (continued).

Property	Optional or mandatory	Description
mfp.db.cloudant.afterWrite.fullCommit	Optional	This property specifies whether an "ensure full commit" operation is used after every write operation to the Cloudant database. The possible values are: `true`, `false`. The default value is `false`.
mfp.db.cloudant.afterWrite.delay	Optional	This property specifies in milliseconds how long to wait after every write operation to the Cloudant database. A value of zero means no wait. The default value is `0`.
mfp.db.cloudant.retry.count	Optional	This property specifies the number of times to retry a Cloudant database query operation until it satisfies the expectations known from the context. The default value is `2`.
mfp.db.cloudant.retry.delay	Optional	This property specifies in milliseconds how long to wait before retrying a Cloudant database query operation. A value of 0 means no wait. The default value is `0`.
mfp.db.cloudant.documentOperation.timeout	Optional	This property specifies in seconds the timeout for the completion of operations on Cloudant documents. A value of zero means an infinite timeout. A negative value means the default value (no override). The default value is 30 seconds.
mfp.db.cloudant.attachmentOperation.timeout	Optional	This property specifies in seconds the timeout for the completion of operations on Cloudant attachments. A value of zero means an infinite timeout. A negative value means the default value (no override). The default value is 600 seconds (10 minutes).

Table 6-30. JNDI properties for Administration Services: licensing.

Property	Optional or mandatory	Description
ibm.worklight.admin.license.key.server.host	• Optional (For Perpetual license) • Mandatory (For Token license)	Host name of the Rational License Key Server.
ibm.worklight.admin.license.key.server.port	• Optional (For Perpetual license) • Mandatory (For Token license)	Port number of the Rational License Key Server.

JNDI properties for MobileFirst Operations Console

The following properties can be set on the web application (`worklightconsole.war`) of MobileFirst Operations Console.

Table 6-31. JNDI properties for the MobileFirst Operations Console

Property	Optional or mandatory	Description
ibm.worklight.admin.endpoint	Optional	Enables the MobileFirst Operations Console to locate the MobileFirst 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.
ibm.worklight.admin.ui.cors.strictssl	Optional	Set to `false` to allow CORS situations where the MobileFirst Operations Console is secured with SSL (HTTPS protocol) while the MobileFirst 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 "JNDI environment entries for MobileFirst projects in production" on page 11-67.

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 MobileFirst Operations Console and Administration Services" on page 15-24.

Related information:

JNDI environment entries for MobileFirst projects in production

Verifying the installation of MobileFirst Server administration

You must log in to the MobileFirst Operations 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 MobileFirst runtime environment that you can manage with the MobileFirst Operations Console, see "Installing the MobileFirst runtime environment."

   Note: If the Application Center and MobileFirst Operations Console are installed in the same Tomcat instance, you cannot log in to the Application Center console and to the MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst Operations Console at the same time. But the Application Center and MobileFirst 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 MobileFirst Operations Console. You can then successfully log in to the MobileFirst Operations Console.

Installing the MobileFirst runtime environment

About this task

For more information about the MobileFirst runtime environment, see "Deploying the project WAR file" on page 11-5.

Installing and configuring for token licensing

If you plan to use token licensing for MobileFirst Server, you must install the Rational Common Licensing library and configure your application server to connect MobileFirst Server to the Rational License Key Server.

The following topics describe the installation overview, the manual installation of Rational Common Licensing library, the configuration of the application server, and the platform limitations for token licensing.

Installation overview for token licensing

The installation process overview for the use of IBM MobileFirst Platform Foundation for iOS with token licensing enabled

About this task

If you intend to use token licensing with IBM MobileFirst Platform Foundation for iOS, make sure that you go through the following preliminary steps in this order.

Important:

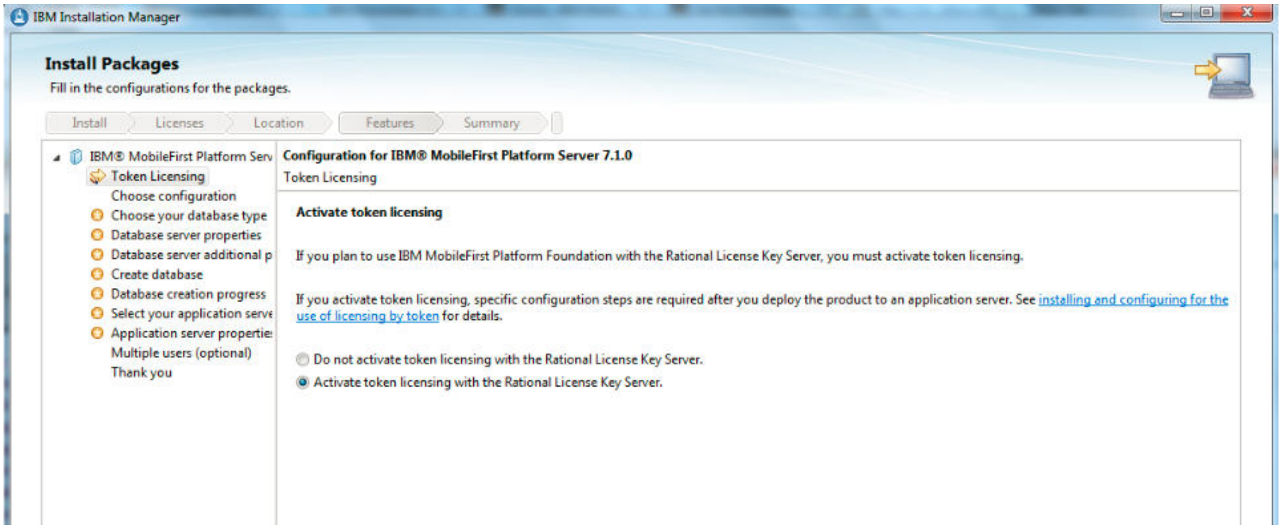
- The ability to activate token licensing is not available in IBM MobileFirst Platform Foundation for iOS V7.1.0 and the interim fixes that are delivered before 14 September 2015. If the graphic mode installation does not display the **Token Licensing** panel, or if the silent installation with `imcl` does not confirm that token licensing is activated, then the token licensing is not activated. The installation is then not compliant with token licensing. To resolve the problem, you need to download IBM MobileFirst Platform Foundation for iOS V7.1.0 from Passport Advantage, or the interim fix 7.1.0.0-MFPF-IF201509132345 dated 15 September 2015.
- Your choice about token licensing (activating it or not) as part of an installation or upgrade that supports token licensing cannot be modified. If later you need to change the token licensing option, you must uninstall IBM MobileFirst Platform Foundation for iOS and reinstall it. If the product is installed with an installer without token licensing support, you can activate the feature at the first upgrade with an interim fix or a release that supports token licensing. The choice to activate token licensing is provided when you make the first upgrade with the interim fix 7.1.0.0-MFPF-IF201509132345 dated 15 September 2015, or a release that has token licensing support.

Procedure

1. Review “Planning for the use of token licensing” on page 6-23 before you start the installation of the product.
2. When you run IBM Installation Manager to install IBM MobileFirst Platform Foundation for iOS, you must activate token licensing. For more information about running IBM Installation Manager, see “Running IBM Installation Manager” on page 6-32. To enable token licensing, do the following steps:

   Graphic mode installation

   If you install the product in graphic mode, select **Activate token licensing with the Rational License Key Server** option in the **Token Licensing** section during the installation.



Install Packages

Fill in the configurations for the packages.

- IBM® MobileFirst Platform Serv
  - Token Licensing
  - Choose configuration
  - Choose your database type
  - Database server properties
  - Database server additional p
  - Create database
  - Database creation progress
  - Select your application serve
  - Application server propertie
  - Multiple users (optional)
  - Thank you

Configuration for IBM® MobileFirst Platform Server 7.1.0

Token Licensing

Activate token licensing

If you plan to use IBM MobileFirst Platform Foundation with the Rational License Key Server, you must activate token licensing.

If you activate token licensing, specific configuration steps are required after you deploy the product to an application server. See installing and configuring for the use of licensing by token for details.

○ Do not activate token licensing with the Rational License Key Server.

● Activate token licensing with the Rational License Key Server.

Figure 6-8. Activating token licensing during the installation of the product.

Silent mode installation

If you install in silent mode, set the value as `true` to the **`user.licensed.by.tokens`** parameter in the response file. For more information, see “Command-line (silent installation) parameters” on page 6-38.

3. Install the MobileFirst Server administration after the product installation is complete. For more information, see “Installing the MobileFirst Server administration” on page 6-47.
4. Configure MobileFirst Server for token licensing. The steps depend on your application server and the details are given for Apache Tomcat, WebSphere Application Server Liberty profile, and WebSphere Application Server full profile. For more information, see “Installing and configuring for token licensing” on page 6-115.

Connecting MobileFirst Server installed on Apache Tomcat to the Rational License Key Server

You must install the Rational Common Licensing native and Java libraries on the Apache Tomcat application server before you connect MobileFirst Server to the Rational License Key Server.

Before you begin

- Rational License Key Server 8.1.4.8 or later must be installed and configured. The network must allow communication to and from MobileFirst Server by opening the two-way communication ports (**`lmrgd`** and **`ibmratl`**). For more information, see Rational License Key Server Portal and How to serve a license key to client machines through a firewall.
- MobileFirst Server must be installed and configured with the option `Activate token licensing with the Rational License Key Server` on your Apache Tomcat as indicated in “Installation overview for token licensing” on page 6-115.

Installing Rational Common Licensing libraries:

Procedure

1. Choose the Rational Common Licensing native library. Depending on your operating system and the bit version of the Java Runtime Environment (JRE) on which your Apache Tomcat is running, you must choose the correct native

library in *product_install_dir*/WorklightServer/tokenLibs/bin/*your_corresponding_platform*/*the_native_library_file*. For example, for Linux x86 with a 64-bit JRE, the library can be found in *product_install_dir*/WorklightServer/tokensLibs/bin/Linux_x86_64/librcl_ibmratl.so.

2. Copy the native library to the computer that runs MobileFirst Server administration service. The directory might be *${CATALINA_HOME}*/bin.

   Note: *${CATALINA_HOME}* is the installation directory of your Apache Tomcat.

3. Copy rcl_ibmratl.jar file to *${CATALINA_HOME}*/lib. The rcl_ibmratl.jar file is a Rational Common Licensing Java library that can be found in *product_install_dir*/WorklightServer/tokenLibs directory. The library uses the native library that is copied in Step 2, and can be loaded only once by Apache Tomcat. This file must be placed in the *${CATALINA_HOME}*/lib directory or any directory in the path of Apache Tomcat common class loader.

   Important: The Java virtual machine (JVM) of Apache Tomcat needs read and execute privileges on the copied native and Java libraries. Both copied files must also be readable and executable at least for the application server process in your operating system.

4. Configure the access to the Rational Common Licensing library by the JVM of your application server. For any operating systems, configure the *${CATALINA_HOME}*/bin/setenv.bat file (or setenv.sh file on UNIX) by adding the following line:

   Windows:

   ```
   set CATALINA_OPTS=%CATALINA_OPTS%
   -Djava.library.path=absolute_path_to_the_previous_bin_directory
   ```

   UNIX:

   ```
   CATALINA_OPTS="$CATALINA_OPTS
   -Djava.library.path=absolute_path_to_the_previous_bin_directory"
   ```

   Note: If you move the configuration folder of the server on which the administration service is running, you must update the **java.library.path** with the new absolute path.

5. Configure MobileFirst Server to access Rational License Key Server. In *${CATALINA_HOME}*/conf/server.xml file, look for the <Context> element of the administration service application, and add in these JNDI configuration lines.

   ```
   <Environment name="ibm.worklight.admin.license.key.server.host" value="rlks_hostname" type="ja
   <Environment name="ibm.worklight.admin.license.key.server.port" value="rlks_port" type="java.l
   ```

   - *rlks_hostname* is the host name of the Rational License Key Server.
   - *rlks_port* is the port of the Rational License Key Server. By default, the value is 27000.

   For more information, see JNDI properties for Administration Services: licensing.

Installing on Apache Tomcat server farm:
About this task

For configuring the connection of MobileFirst Server on Apache Tomcat server farm, you must follow all the steps that are described in “Installing Rational Common Licensing libraries” on page 6-116 for each node of your server farm where the MobileFirst Server administration service is running. For more information about server farm, see “Installing a server farm” on page 6-127.

Connecting MobileFirst Server installed on WebSphere Application Server Liberty profile to the Rational License Key Server

You must install the Rational Common Licensing native and Java libraries on the Liberty profile before you connect MobileFirst Server to the Rational License Key Server.

Before you begin

- Rational License Key Server 8.1.4.8 or later must be installed and configured. The network must allow communication to and from MobileFirst Server by opening the two-way communication ports (**`lmrgd`** and **`ibmratl`**). For more information, see Rational License Key Server Portal and How to serve a license key to client machines through a firewall.
- You must have WebSphere Application Server Liberty profile 8.5.5.x. WebSphere Application Server Liberty profile 8.5.0.x is not supported.
- MobileFirst Server must be installed and configured with the option `Activate token licensing with the Rational License Key Server` on your Liberty profile as indicated in "Installation overview for token licensing" on page 6-115.

Installing Rational Common Licensing libraries:

Procedure

1. Define a shared library for the Rational Common Licensing client. This library uses native code and can be loaded only once by the application server. Thus, the applications that use it must reference it as a common library.
   a. Choose the Rational Common Licensing native library. Depending on your operating system and the bit version of the Java Runtime Environment (JRE) on which your Liberty profile is running, you must choose the correct native library in *`product_install_dir`*`/WorklightServer/tokenLibs/bin/`*`your_corresponding_platform/the_native_library_file`*. For example, for Linux x86 with a 64-bits JRE, the library can be found in *`product_install_dir`*`/WorklightServer/tokensLibs/bin/Linux_x86_64/librcl_ibmratl.so`.
   b. Copy the native library to the computer that runs MobileFirst Server administration service. The directory might be `${`*`shared.resource.dir`*`}/rcllib`. For more information about standard location of *${shared.resource.dir}*, see WebSphere Application Server Liberty Core - Directory locations and properties. If the `rcllib` folder does not exist, create this folder and then copy the native library file over.

      Note: Ensure that the Java virtual machine (JVM) of the application server has both read and execute privileges on the native library. On Windows, the following exception appears in the application server log if the JVM of the application server does not have the executable rights on the copied native library.

      ```
      com.ibm.rcl.ibmratl.LicenseConfigurationException: java.lang.UnsatisfiedLinkError: rcl_ibmratl
      ```
   c. Copy `rcl_ibmratl.jar` file to `${`*`shared.resource.dir`*`}/rcllib`. The `rcl_ibmratl.jar` file is a Rational Common Licensing Java library that can be found in *`product_install_dir`*`/WorklightServer/tokenLibs` directory.

      Note: The Java virtual machine (JVM) of Liberty profile must have the privilege to read the copied Java library. This file must also have readable privilege (at least for the application server process) in your operating system.

d. Declare a shared library that uses the `rcl_ibmratl.jar` file in the `${server.config.dir}/server.xml` file.

```
<!-- Declare a shared Library for the RCL client. -->
<!- This library can be loaded only once because it uses native code. -->
<library id="RCLLibrary">
  <fileset dir="${shared.resource.dir}/rcllib" includes="rcl_ibmratl.jar"/>
</library>
```

e. Declare the shared library as a common library for the MobileFirst Server administration service application by adding an attribute (**`commonLibraryRef`**) to the class loader of the application. As the library can be loaded only once, it must be used as a common library, and not as a private library.

```
<application id="wladmin" name="wladmin" location="worklightadmin.war" type="war">
  [...]
  <!- Declare the shared library as an attribute commonLibraryRef to the class loader of th
  <classloader delegation="parentLast" commonLibraryRef="RCLLibrary">
    <commonLibrary id="worklightlib_wladmin">
      <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.jar'
    </commonLibrary>
  </classloader>
</application>
```

f. Configure the access to the Rational Common Licensing library by the JVM of your application server. For any operating systems, configure the *`${wlp.user.dir}`*`/servers/`*`server_name`*`/jvm.options` file by adding the following line:

```
-Djava.library.path=Absolute_path_to_the_previously_created_rcllib_folder
```

Note: If you move the configuration folder of the server on which the administration service is running, you must update the **`java.library.path`** with the new absolute path.

2. Configure MobileFirst Server to access Rational License Key Server.

   In the `${server.config.dir}/server.xml` file, add these JNDI configuration lines.

```
<jndiEntry jndiName="ibm.worklight.admin.license.key.server.host" value="rlks_hostname"/>
<jndiEntry jndiName="ibm.worklight.admin.license.key.server.port" value="rlks_port"/>
```

- *rlks_hostname* is the host name of the Rational License Key Server.
- *rlks_port* is the port of the Rational License Key Server. By default, the value is 27000.

For more information, see JNDI properties for Administration Services: licensing.

Installing on Liberty profile server farm:
About this task

For configuring the connection of MobileFirst Server on Liberty profile server farm, you must follow all the steps that are described in Installing Rational Common Licensing libraries for each node of your server farm where the MobileFirst Server administration service is running. For more information about server farm, see "Installing a server farm" on page 6-127.

Connecting MobileFirst Server installed on WebSphere Application Server to the Rational License Key Server

You must configure a shared library for the Rational Common Licensing libraries on WebSphere Application Server before you connect MobileFirst Server to the Rational License Key Server.

Before you begin

- Rational License Key Server 8.1.4.8 or later must be installed and configured. The network must allow communication to and from MobileFirst Server by opening the two-way communication ports (**`lmrgd`** and **`ibmratl`**). For more information, see Rational License Key Server Portal and How to serve a license key to client machines through a firewall.
- MobileFirst Server must be installed and configured with the option `Activate token licensing with the Rational License Key Server` on your WebSphere Application Server as indicated in "Installation overview for token licensing" on page 6-115.

Installing Rational Common Licensing library on a stand-alone server: Procedure

1. Define a shared library for the Rational Common Licensing library. This library uses native code and can be loaded only once by a class loader during the application server lifecycle. For this reason, the library is declared as a shared library and associated to all the application servers that run the MobileFirst Server administration service. For more information about the reasons to declare this library as a shared library, see Configuring native libraries in shared libraries.
   a. Choose the Rational Common Licensing native library. Depending on your operating system and the bit version of the Java Runtime Environment (JRE) on which your WebSphere Application Server is running, you must choose the correct native library in *product_install_dir*/WorklightServer/tokenLibs/bin/*your_corresponding_platform*/*the_native_library_file*.

      For example, for Linux x86 with a 64-bits JRE, the library can be found in *product_install_dir*/WorklightServer/tokensLibs/bin/Linux_x86_64/librcl_ibmratl.so.

      To determine the bit version of the Java Runtime Environment for a stand-alone WebSphere Application Server or WebSphere Application Server Network Deployment installation, run the `versionInfo.bat` on Windows or `versionInfo.sh` on UNIX from the bin directory. The `versionInfo.sh` file is in `/opt/IBM/WebSphere/AppServer/bin`. Look at the Architecture value in the **Installed Product** section. The Java Runtime Environment is 64-bit if the Architecture value mentions it explicitly or if it is suffixed with 64 or _64.

   b. Place the native library that corresponds to your platform in a folder of your operating system. For example, `/opt/IBM/RCL_Native_Library/`.

   c. Copy `rcl_ibmratl.jar` file to `/opt/IBM/RCL_Native_Library/`. The `rcl_ibmratl.jar` file is a Rational Common Licensing Java library that can be found in *product_install_dir*/WorklightServer/tokenLibs directory.

      Important: The Java virtual machine (JVM) of the application server needs read and execute privileges on the copied native and Java libraries. Both copied files must also be readable and executable at least for the application server process in your operating system.

   d. Declare a shared library in WebSphere Application Server administrative console.
      1) Log in to WebSphere Application Server administrative console.
      2) Expand **Environment** > **Shared Libraries**.
      3) Select a scope that is visible by all servers that run the MobileFirst Server administration service. For example, a cluster.
      4) Click **New**.

5) Enter a name for the library in the **Name** field. For example, RCL Shared Library.

6) In the **Classpath** field, enter the path to the rcl_ibmratl.jar file. For example, /opt/IBM/RCL_Native_Library/rcl_ibmratl.jar.

7) Click **OK** and save the changes. This setting takes effect when the server is restarted.

e. Associate the shared library with all servers that run the MobileFirst Server administration service.

Associating the shared library to a server allows the shared library to be used by several applications. If you need the Rational Common Licensing client only for the MobileFirst Server administration service, you can create a shared library with an isolated class loader and associate it with the administration service application.

The following instruction is to associate the library with a server. For WebSphere Application Server Network Deployment, you must complete this instruction for all the servers that run the MobileFirst Server administration service.

1) Set the class loader policy and mode.

a) In WebSphere Application Server administrative console, click **Servers** > **Server Types** > **WebSphere application servers** > *server_name* to access the application server setting page.

b) Set the values for the application class-loader policy and class loading mode of the server:

- **Classloader policy**: Multiple
- **Class loading mode**: Classes loaded with parent class loader first

c) In **Server Infrastructure** section, click **Java and Process Management** > **Class loader**.

d) Click **New** and ensure that class loader order is set to Classes loaded with parent class loader first.

e) Click **Apply** to create a new class loader ID.

2) Create a library reference for each shared library file that your application needs.

a) Click the name of the class loader that is created in the previous step.

b) In **Additional properties** section, click **Shared library references**.

c) Click **Add**.

d) At the **Library reference settings** page, select the appropriate library reference. The name identifies the shared library file that your application uses. For example, RCL Shared Library.

e) Click **Apply** and then save the changes.

2. Configure the environment entries for MobileFirst Server administration service web application.

a. In WebSphere Application Server administrative console, click **Applications** > **Application Types** > **WebSphere enterprise applications** and select the administration service application: Worklight_Administration_Service.

b. In **Web Module Properties** section, click **Environment entries for web modules**.

c. Enter the values for **ibm.worklight.admin.license.key.server.host** and **ibm.worklight.admin.license.key.server.port**.

- **`ibm.worklight.admin.license.key.server.host`** is the host name of the Rational License Key Server.
- **`ibm.worklight.admin.license.key.server.port`** is the port of the Rational License Key Server. By default, the value is 27000.

d. Click **OK** and save the changes.

3. Configure the access to the Rational Common Licensing library by the application server JVM.

   a. In WebSphere Application Server administrative console, click **Servers** > **Server Types** > **WebSphere Application Servers** and select your server.

   b. In **Server Infrastructure** section, click **Java and Process Management** > **Process Definition** > **Java Virtual Machine** > **Custom Properties** > **New** to add a custom property.

   c. In the **Name** field, type the name of the custom property as `java.library.path`.

   d. In the **Value** field, enter the path of the folder where you place the native library file in Step 1b. For example, `/opt/IBM/RCL_Native_Library/`.

   e. Click **OK** and save the changes.

4. Restart your application server.

Installing Rational Common Licensing library on WebSphere Application Server Network Deployment:
About this task

For installing the native library on a WebSphere Application Server Network Deployment, you must follow all the steps that are described in "Installing Rational Common Licensing library on a stand-alone server" on page 6-120. The servers or clusters that you configure must be restarted in order for the changes to take effect.

Each node of your WebSphere Application Server Network Deployment must have a copy of the Rational Common Licensing native library.

Each server where the MobileFirst Server administration service runs must be configured to have access to the native library copied on your local computer. These servers must also be configured to connect to Rational License Key Server.

Important:

- If you use a cluster with WebSphere Application Server Network Deployment, your cluster can change. You must configure each newly added server in your cluster, where the administration services are running.
- Asymmetric deployment with runtimes and administration services in different server or cluster is not supported on WebSphere Application Server Network Deployment. For more information, see "Planning for the use of token licensing" on page 6-23.

Limitations of supported platforms for token licensing

The list of operating system, its version, and the hardware architecture that supports MobileFirst Server with token licensing enabled.

For token licensing, the MobileFirst Server needs to connect to the Rational License Key Server by using the Rational Common Licensing library.

This library is composed of a Java library and also native libraries. These native libraries depend on the platform where MobileFirst Server is running. Thus, the token licensing by MobileFirst Server is supported only on platforms where the Rational Common Licensing library can be run.

The following table describes the platforms that support MobileFirst Server with the token licensing.

Table 6-32. Supported Operating System, Operating System version, and hardware architecture.

Operating System	Operating System version	Hardware architecture
AIX	6.1	POWER7® (64-bit only)
AIX	7.1	POWER7 (64-bit only)
AIX	7.1	POWER8® (64-bit only)
Red Hat Server Editions	6	x86-64 only
Solaris	10	SPARC (64-bit only)
SUSE Linux Enterprise Server	11	x86-64 only
Windows Server	2012	x86-64 only

Token licensing does not support 32-bit Java Runtime Environment (JRE). Make sure that the application server uses a 64-bit JRE.

If you intend to use token licensing with WebSphere Application Server Liberty profile, make sure that the version is 8.5.5.x. WebSphere Application Server Liberty profile 8.5.0.x is not supported.

Troubleshooting token licensing problems

Find information to help resolve issues that you might encounter with token licensing if you activated this feature when you installed MobileFirst Server.

When you start the MobileFirst Server administration services after you complete "Installing and configuring for token licensing" on page 6-115, some errors or exceptions can be emitted in the application server log or on MobileFirst Operations Console. These exceptions might be due to incorrect installation of the Rational Common Licensing library and configuration of the application server.

Apache Tomcat
Check `catalina.log` or `catalina.out` file, depending on your platform.

WebSphere Application Server Liberty profile
Check `messages.log` file.

WebSphere Application Server full profile
Check `SystemOut.log` file.

Important: If token licensing is installed on WebSphere Application Server Network Deployment or a cluster, you must check the log of each server.

Here is a list of exceptions that might occur after the installation and configuration for token licensing:

- "Rational Common Licensing native library is not found"
- "Rational Common Licensing shared library is not found" on page 6-125
- "The Rational License Key Server connection is not configured" on page 6-125
- "The Rational License Key Server is not accessible" on page 6-126
- "Failed to initialize Rational Common Licensing API" on page 6-126
- "Insufficient token licenses" on page 6-127
- "Invalid rcl_ibmratl.jar file" on page 6-127

Rational Common Licensing native library is not found

```
FWLSE3125E: The Rational Common Licensing native library is not found. Make
sure the JVM property (java.library.path) is defined with the right path
and the native library can be executed. Restart IBM MobileFirst Platform
Server after taking corrective action.
```

For WebSphere Application Server full profile

Possible causes to this error might be:

- No common property with name **java.library.path** is defined at server level.
- The path that is given as the value for the **java.library.path** property does not contain the Rational Common Licensing native library.
- The native library does not have appropriate permissions. The library must have the read and execute privileges on UNIX and Windows for the user who accesses it with the Java Runtime Environment of the application server.

For WebSphere Application Server Liberty profile and Apache Tomcat

Possible causes to this error might be:

- The path to the Rational Common Licensing native library given as the value of **java.library.path** property is either not set or incorrect.
  - For Liberty profile, check `${wlp.user.dir}/servers/server_name/jvm.options` file.
  - For Apache Tomcat, check `${CATALINA_HOME}/bin/setenv.bat` file or `setenv.sh` file, depending on your platform.
- The native library is not found in the path that is defined to the **java.library.path** property. Check that the native library exists in the defined path with the expected name.
- The native library does not have appropriate permissions. The error might be preceded by this exception:
  ```
  com.ibm.rcl.ibmratl.LicenseConfigurationException:
  java.lang.UnsatisfiedLinkError: {0}\rcl_ibmratl.dll: Access is
  denied
  ```

  The Java Runtime Environment of the application server needs read and execute privileges on this native library. The library file must also be readable and executable at least for the application server process in your operating system.
- The shared library that uses the `rcl_ibmratl.jar` file is not defined in the `${server.config.dir}/server.xml` file for Liberty profile. The `rcl_ibmratl.jar` might also not in the correct directory or the directory does not have the appropriate permissions.

- The shared library that used the rcl_ibmratl.jar file is not declared as a common library for the MobileFirst Server administration service application in the ${server.config.dir}/server.xml file for the Liberty profile.
- There is a mix of 32-bit and 64-bit objects between the Java Runtime Environment of the application server and the native library. For example, a 32-bit Java Runtime Environment is used with a 64-bit native library. This mix is not supported.

Rational Common Licensing shared library is not found

```
FWLSE3126E: The Rational Common Licensing shared library is not found. Make
sure the shared library is configured. Restart IBM MobileFirst Platform
Server after taking corrective action.
```

Possible causes to this error might be:

- The rcl_ibmratl.jar file is not in the expected directory.
  - For Apache Tomcat, check that this file is in ${CATALINA_HOME}/lib directory.
  - For WebSphere Application Server Liberty profile, check that this file is in the directory as defined in the server.xml file for the shared library of the Rational Common Licensing client. For example, ${shared.resource.dir}/rcllib. In the server.xml file, ensure that this shared library is correctly referenced as a common library for MobileFirst Server administration service application.
  - For WebSphere Application Server, make sure that this file is in the directory that is specified in the class path of the WebSphere Application Server shared library. Check that the class path of that shared library contains this entry: *absolute_path*/rcl_ibmratl.jar whereas *absolute_path* is the absolute path of the rcl_ibmratl.jar file.
- The **java.library.path** property is not set for the application server. Define a property with name **java.library.path** and set the path to the Rational Common Licensing native library as the value. For example, /opt/IBM/RCL_Native_Library/.
- The native library does not have the expected permissions. On Windows, the Java Runtime Environment of the application server must have the read and executable rights on the native library.
- There is a mix of 32-bit and 64-bit objects between the Java Runtime Environment of the application server and the native library. For example, a 32-bit Java Runtime Environment is used with a 64-bit native library. This mix is not supported.

The Rational License Key Server connection is not configured

```
FWLSE3127E: The Rational License Key Server connection is not configured.
Make sure the admin JNDI properties
"ibm.worklight.admin.license.key.server.host" and
"ibm.worklight.admin.license.key.server.port" are set. Restart IBM
MobileFirst Platform Server after taking corrective action.
```

Possible causes to this error might be:

- The Rational Common Licensing native library and the shared library that uses the rcl_ibmratl.jar file are correctly configured but the value of JNDI properties (**ibm.worklight.admin.license.key.server.host** and **ibm.worklight.admin.license.key.server.port**) is not set in the MobileFirst Server administration services application.

- The Rational License Key Server is down.
- The host computer on which Rational License Key Server is installed cannot be reached. Check the IP address or host name with the specified port.

The Rational License Key Server is not accessible

```
FWLSE3128E: The Rational License Key Server "{port}@{IP address or
hostname}" is not accessible. Make sure that license server is running and
accessible to IBM MobileFirst Platform Server. If this error occurs at
runtime startup, restart IBM MobileFirst Platform Server after taking
corrective action.
```

Possible causes to this error might be:

- The Rational Common Licensing shared library and the native library are correctly defined but there is no valid configuration to connect to the Rational License Key Server. Check the IP address, the host name, and the port of the license server. Make sure that the license server is started and accessible from the computer where the application server is installed.
- The native library is not found in the path that is defined to the **`java.library.path`** property.
- The native library does not have appropriate permissions.
- The native library is not in the defined directory.
- The Rational License Key Server is behind a firewall. The error might be preceded by this exception: `[ERROR] Failed to get license for application 'WorklightStarter' because Rational Licence Key Server ({port}@{IP address or hostname}) is either down or not accessible com.ibm.rcl.ibmratl.LicenseServerUnreachableException. All license files searched for features: {port}@{IP address or hostname}`

  Ensure that the license manager daemon (**`lmgrd`**) port and the vendor daemon (**`ibmratl`**) port are open in your firewall. For more information, see How to serve a license key to client machines through a firewall.

Failed to initialize Rational Common Licensing API

```
Failed to initialize Rational Common Licensing (RCL) API because its native
library could not be found or loaded
com.ibm.rcl.ibmratl.LicenseConfigurationException:
java.lang.UnsatisfiedLinkError: rcl_ibmratl (Not found in
java.library.path)
```

Possible causes to this error might be:

- The Rational Common Licensing native library is not found in the path that is defined to the **`java.library.path`** property. Check that the native library exists in the defined path with the expected name.
- The **`java.library.path`** property is not set for the application server. Define a property with name **`java.library.path`** and set the path to the Rational Common Licensing native library as the value. For example, `/opt/IBM/RCL_Native_Library/`.
- There is a mix of 32-bit and 64-bit objects between the Java Runtime Environment of the application server and the native library. For example, a 32-bit Java Runtime Environment is used with a 64-bit native library. This mix is not supported.

Insufficient token licenses

```
FWLSE3129E: Insufficient token licenses for feature "{0}".
```

This error occurs when the remaining number of token licenses on the Rational License Key Server is not enough to deploy a new MobileFirst application.

Invalid `rcl_ibmratl.jar` file

```
UTLS0002E: The shared library RCL Shared Library contains a classpath entry
which does not resolve to a valid jar file, the library jar file is
expected to be found at {0}/rcl_ibmratl.jar.
```

Note: For WebSphere Application Server and WebSphere Application Server Network Deployment only
Possible causes to this error might be:

- The `rcl_ibmratl.jar` Java library does not have the appropriate permissions. The error might be followed by another exception: `java.util.zip.ZipException: error in opening zip file`.

  Check that the `rcl_ibmratl.jar` file has the read permission for the user who installs WebSphere Application Server.
- If there is no other exception, the `rcl_ibmratl.jar` file that is referenced in the class path of the shared library might be invalid or does not exist. Check that the `rcl_ibmratl.jar` file is valid or exists in the defined path.

Related links

"Connecting MobileFirst Server installed on Apache Tomcat to the Rational License Key Server" on page 6-116
You must install the Rational Common Licensing native and Java libraries on the Apache Tomcat application server before you connect MobileFirst Server to the Rational License Key Server.

"Connecting MobileFirst Server installed on WebSphere Application Server Liberty profile to the Rational License Key Server" on page 6-118
You must install the Rational Common Licensing native and Java libraries on the Liberty profile before you connect MobileFirst Server to the Rational License Key Server.

"Connecting MobileFirst Server installed on WebSphere Application Server to the Rational License Key Server" on page 6-119
You must configure a shared library for the Rational Common Licensing libraries on WebSphere Application Server before you connect MobileFirst Server to the Rational License Key Server.

Installing a server farm

MobileFirst 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.

When you intend to plan a server farm installation, you should first see "Planning deployment of administration components and runtimes" on page 6-8, and in particular see "Server farm topology" on page 6-11.

In IBM MobileFirst Platform Foundation for iOS, 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. MobileFirst 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:

- MobileFirst Server is installed on multiple Tomcat application servers.
- MobileFirst Server is installed on multiple WebSphere Application Server servers but not on WebSphere Application Server Network Deployer.
- MobileFirst Server 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.

Why it is mandatory to declare a farm

Each time a management operation is performed through the MobileFirst Operations Console or through the MobileFirst Administration Services application, the operation needs to be replicated to all instances of a runtime environment. Examples of such management operations are the uploading of a new version of a wlapp or of an adapter. The replication is done via JMX calls performed by the MobileFirst Administration Services application instance that handles the operation. The Administration Service needs to contact all runtime instances in the cluster. In environments listed under “When to declare a server farm,” the runtime can be contacted through JMX only if a farm is configured. If a server is added to a cluster without proper configuration of the farm, the runtime in that server will be in an inconsistent state after each management operation, and until it is restarted again.

Configuring a server farm

You must configure each server in the farm according to the requirements of the single type of application server used for each member of the server farm.

About this task

When you plan a server farm, first create stand-alone servers that communicate with the same database instance. Then, modify the configuration of these servers to make them members of a server farm.

Procedure

1. Choose the type of application server to use to configure the members of the server farm. IBM MobileFirst Platform Foundation for iOS supports these application servers in server farms:
   - WebSphere Application Server full profile.

     Note: In a farm topology, you cannot use the RMI JMX connector. In this topology, IBM MobileFirst Platform Foundation for iOS supports only the SOAP connector.
   - WebSphere Application Server Liberty profile.

- Apache Tomcat.

To know which versions of application servers are supported, see "System requirements" on page 2-9.

Note: IBM MobileFirst Platform Foundation for iOS supports only homogeneous server farms. A server farm is homogeneous when it connects application servers of the same type. Attempting to associate different types of application servers might lead to unpredictable behavior at run time. For example, a farm with a mix of Apache Tomcat servers and WebSphere Application Server full profile servers is an invalid configuration.

2. Decide which database that you want to use. You can choose from:
   - DB2
   - MySQL
   - Oracle

   MobileFirst databases are shared between the application servers in a farm, which means:
   - You create the database only once, whatever the number of servers in the farm.
   - You cannot use the Derby database in a farm topology, because the Derby database allows only a single connection at a time.

   For more information about databases, see "Planning the creation of the databases" on page 6-19.
3. Set up as many stand-alone servers as the number of members that you want in the farm. Each of these stand-alone servers must communicate with the same database. You must make sure that any port used by any of these servers is not also used by another server configured on the same host. This constraint applies to the ports used by http, https, REST, SOAP, and RMI protocols.

   To set up the servers, you can choose one of the following methods of installation:
   - Install them by using the Server Configuration Tool.
   - Install them by using configuration Ant scripts.
   - Install them manually.

   Each of these servers must have the MobileFirst Administration Services and one or more MobileFirst runtime environments deployed on it.

   For more information about setting up a server, see "Planning the installation of MobileFirst Server" on page 6-3.

   When each of these servers is working properly in a stand-alone topology, you can transform them into nodes of a server farm.
4. Stop all the servers that are intended to become members of the farm.
5. Configure each server appropriately for the type of application server. You must set some JNDI properties correctly. For Apache Tomcat, you must also check that the JVM arguments are properly defined.
   - **WebSphere Application Server Liberty profile**

     In the `server.xml` file, set the JNDI properties shown in the following sample code.

     ```
     <jndiEntry jndiName="ibm.worklight.topology.clustermode" value="Farm"/>
     <jndiEntry jndiName="ibm.worklight.admin.serverid" value="farm_member_1"/>
     <jndiEntry jndiName="ibm.worklight.admin.jmx.user" value="myRESTConnectorUser"/>
     <jndiEntry jndiName="ibm.worklight.admin.jmx.pwd" value="password-of-rest-connector-user"/>
     <jndiEntry jndiName="ibm.worklight.admin.jmx.host" value="93.12.0.12"/>
     ```

     These properties must be set with appropriate values:

  - **ibm.worklight.admin.serverid**: The identifier that you defined for this farm member. This identifier must be unique across all farm members.
  - **ibm.worklight.admin.jmx.user** and **ibm.worklight.admin.jmx.pwd**: These values must match the credentials of a user as declared in the `<administrator-role/>` element.
  - **ibm.worklight.admin.jmx.host**: Set this parameter to the IP or the host name that is used by remote members to access this server. Therefore, do not set it to `localhost`. This host name is used by the other members of the farm and must be accessible to all farm members.
  - **ibm.worklight.admin.jmx.port**: Set this parameter to the server HTTPS port that is used for the JMX REST connection. You can find the value in the `<httpEndpoint>` element of the `server.xml` file.

- **Apache Tomcat**

  Modify the `conf/server.xml` file to set the following JNDI properties in the MobileFirst administration context and in every MobileFirst runtime context.

  ```
  <Environment name="ibm.worklight.topology.clustermode" value="Farm" type="java.lang.String" ove
  <Environment name="ibm.worklight.admin.serverid" value="farm_member_1" type="java.lang.String"
  ```

  The **ibm.worklight.admin.serverid** property must be set to the identifier that you defined for this farm member. This identifier must be unique across all farm members.

  You must make sure that the `-Djava.rmi.server.hostname` JVM argument is set to the IP or the host name that is used by remote members to access this server. Therefore, do not set it to `localhost`. This argument is set in the *CATALINA_OPTS* environment variable.

- **WebSphere Application Server full profile**

  You must declare the following environment variables in the MobileFirst Administration Services and in every MobileFirst runtime application deployed on the server.

  - *ibm.worklight.topology.clustermode*
  - *ibm.worklight.admin.serverid*

  a. In the WebSphere Application Server console, select **Applications** > **Application Types** > **WebSphere Enterprise applications**.

  b. Select the MobileFirst Administration Service application.

  c. In Web Module Properties, click **Environment entries for Web Modules** to display the JNDI properties.

  d. Set the values of the following properties.

     - Set **ibm.worklight.topology.clustermode** to `Farm`.
     - Set **ibm.worklight.admin.serverid** to the identifier that you chose for this farm member.
     - Set **ibm.worklight.admin.jmx.user** to a user name that has access to the SOAP connector.
     - Set **ibm.worklight.admin.jmx.pwd** to the password of the user declared in **ibm.worklight.admin.jmx.user**.

  e. Verify that **ibm.worklight.admin.jmx.connector** is set to `SOAP`.

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

  g. Make similar changes for every MobileFirst runtime application deployed on the server.

6. Exchange the server certificates in their truststores. Exchanging the server certificates in their truststores is mandatory for farms that use WebSphere

Application Server full profile and WebSphere Application Server Liberty profile, because in these farms communications between the servers is secured by SSL.

- **WebSphere Application Server Liberty profile**

  You can configure the truststore by using IBM utilities such as Keytool or iKeyman.

  – For more information about Keytool, see Keytool in the IBM SDK, Java Technology Edition.

  – For more information about iKeyman, see iKeyman in the IBM SDK, Java Technology Edition.

  The locations of keystore and truststore are defined in the `server.xml` file. See the `keyStoreRef` and `trustStoreRef` attributes in SSL configuration attributes. By default, the keystore of Liberty profile is at `${server.config.dir}/resources/security/key.jks`. If the truststore reference is missing or not defined in the `server.xml` file, the keystore that is specified by `keyStoreRef` is used. The server uses the default keystore and the file is created the first time that the server runs. In that case, a default certificate is created with a validity period of 365 days. For production, you might consider using your own certificate (including the intermediate ones if needed) or changing the expiration date of the generated certificate.

  Note: If you want to confirm the location of the truststore, you can do so by adding the following declaration to the `server.xml` file:

  ```
  <logging traceSpecification="SSL=all:SSLChannel=all"/>
  ```

  Then, start the server and look for lines that contain `com.ibm.ssl.trustStore` in the `${wlp.install.dir}/usr/servers/`*`server_name`*`/logs/trace.log` file.

a. Import the certificates of the other servers in the farm into the truststore that is referenced by the `server.xml` configuration file of the server.

   If you are using the Keytool utility, do the following steps for each server of the farm:

   1) In the *`${wlp.user.dir}`*`/servers/`*`server_name`*`/resources/security` directory, enter the command: **`keytool -list -keystore key.jks`**.

      This command shows the certificates in the keystore and their corresponding alias names. You are prompted for the password of the keystore before you can see the truststore entries. This is the case for all the next commands with Keytool utility. If you do not use a custom keystore, the server must be started once to generate a default truststore `key.jks` in the mentioned directory. The password of the default keystore is `worklight`.

      Note: If you replace the default `key.jks` by your own truststore, you might have to append the option **`-storetype`** to every Keytool command you enter to match your truststore type.

   2) Export all the certificates in the truststore with the command: **`keytool -exportcert -keystore key.jks -alias`** *`cert_entry`* **`-file`** *`cert_entry`*`.cert` where *cert_entry* must be replaced with each certificate alias name.

   3) Copy all the extracted certificates to the other servers of the farm so that they can be imported in their respective truststores. This step assumes that you copy them in the remote directories: *`${wlp.user.dir}`*`/servers/`*`server_name`*`/resources/security`.

   4) Import the certificates in the truststore of the other servers of the farm.

On each server, change the directory to `${wlp.user.dir}/servers/server_name/resources/security`. Import each certificate in the truststore in turn by using the command: **`keytool -importcert -keystore key.jks -alias`** *`new_alias_name`* **`-file`** *`cert_entry`*`.cert` where *`cert_entry`*`.cert` is to be replaced with the name of each certificate that you copied at previous step. The *new_alias_name* is the name of the entry that is created in the truststore to store this certificate. You are prompted to explicitly trust the certificate.

Note: Each alias in the truststore must be unique. You are warned by the tool if a certificate is already present but stored with a different alias.

b. Restart each instance of WebSphere Application Server Liberty profile to make the security configuration take effect.

The following steps are required for single sign-on (SSO) to work:

c. Start one member of the farm if not already done. In the default LTPA configuration, after the Liberty server starts successfully, it generates an LTPA keystore as `${wlp.install.dir}/usr/servers/server_name/resources/security/ltpa.keys`.

d. Copy the `ltpa.keys` file to the `${wlp.install.dir}/usr/servers/server_name/resources/security` directory of each farm member to replicate the LTPA keystores across the farm members.

e. Start the other members of the farm.

For more information about LTPA configuration, see Configuring LTPA on the Liberty profile.

- **WebSphere Application Server full profile**

  Configure the truststore in the WebSphere Application Server administration console.

a. Log in to WebSphere Application Server administration console.

b. Select **Security** > **SSL certificate and key management**.

c. In Related Items, select **Keystores and certificates**.

d. In the **Keystore usages** field, make sure that **SSL keystores** is selected. You can now import the certificates from all the other servers in the farm.

e. Click **NodeDefaultTrustStore**.

f. In **Additional Properties**, select **Signer certificates**.

g. Click **Retrieve from port**. You can now enter communication and security details of each of the other servers in the farm. Follow the next steps for each of the other farm members.

h. In the **Host** field, enter the server host name or IP address.

i. In the **Port** field, enter the HTTPS transport (SSL) port.

j. In "SSL configuration for outbound connection", select **NodeDefaultSSLSettings**.

k. In the **Alias** field, enter an alias for this signer certificate.

l. Click **Retrieve signer information**.

m. Review the information that is retrieved from the remote server and then click **OK**.

n. Click **Save**.

o. Restart the server.

What to do next

If you are creating a farm whose members use WebSphere Application Server Liberty profile, you can now set up an IBM HTTP Server for Liberty. For more information, see "Setting up HTTP Server in a WebSphere Application Server Liberty profile farm" on page 6-312.

Verifying a farm configuration

To verify a server farm configuration, start all the servers, deploy an application to one of the servers of the farm, and then check the log files of each server to confirm that all servers have been updated.

About this task

The purpose of this task is to verify that a farm is configured properly and that administration operations are propagated on all servers of a farm.

Procedure

1. Start all the servers of the farm.
2. Deploy a MobileFirst application to one of the servers of the farm. You can use the MobileFirst Operations Console or the `wladm` program with the `deploy app` command. For more information about the `deploy app` command, see "Commands for apps" on page 12-45.
3. Once the operation is completed, review the log file of the application server and search for the entries of class BaseTransaction. Verify that all servers have been updated.

   This is the example of a Liberty log file:

   ```
   $ grep BaseTransaction messages.log

   [9/22/14 1:03:17:032 CEST] 0000006d com.ibm.worklight.admin.actions.BaseTransaction
   [9/22/14 1:03:17:251 CEST] 0000006e com.ibm.worklight.admin.actions.BaseTransaction
   [9/22/14 1:03:19:123 CEST] 00000072 com.ibm.worklight.admin.actions.BaseTransaction
   [9/22/14 1:03:19:123 CEST] 00000072 com.ibm.worklight.admin.actions.BaseTransaction
   [9/22/14 1:03:19:138 CEST] 00000071 com.ibm.worklight.admin.actions.BaseTransaction
   [9/22/14 1:03:19:138 CEST] 00000071 com.ibm.worklight.admin.actions.BaseTransaction
   ```

   Troubleshooting

   - If you use an environment ID (see "List of JNDI properties for MobileFirst Server administration" on page 6-101 and "JNDI environment entries for MobileFirst projects in production" on page 11-67), for each server, verify that the **environmentId** value for the MobileFirst Administration Service is the same as the **environmentId** value for the runtime.
   - If you have several servers on the same computer, you must verify that all servers are using a different JMX port:
     - For WebSphere Application Server, the port that is used for JMX is the SOAP port.
     - For WebSphere Application Server Liberty, the port that is used for JMX is the HTTPS port.
     - For Apache Tomcat, the port that is used for JMX is the RMI port, which is specified in the Ant tasks or in the `setenv` file for a manual installation.

     If the JMX port is not available for a server, the MobileFirst runtime environment cannot start.

Lifecycle of a server farm node

You can configure heartbeat rate and timeout values to indicate possible server problems among farm members by triggering a change in status of an affected node.

When a server configured as a farm node is started, the Administration Service on that server automatically registers it as a new farm member.

When a farm member is shut down, it automatically unregisters from the farm.

A heartbeat mechanism exists to keep track of farm members that might become unresponsive, for example, because of a power outage or a server failure. In this heartbeat mechanism, MobileFirst runtimes send periodically at a given rate a heartbeat to MobileFirst administration services. If the MobileFirst administration services register that too long a time has elapsed since a farm member sent a heartbeat, then the farm member is considered to be down.

Farm members considered to be down do not serve any more requests to mobile applications.

Having one or more nodes down does not prevent the other members of the farm from:

- Serving requests correctly to mobile applications
- Accepting new management operations triggered through MobileFirst Operations Console

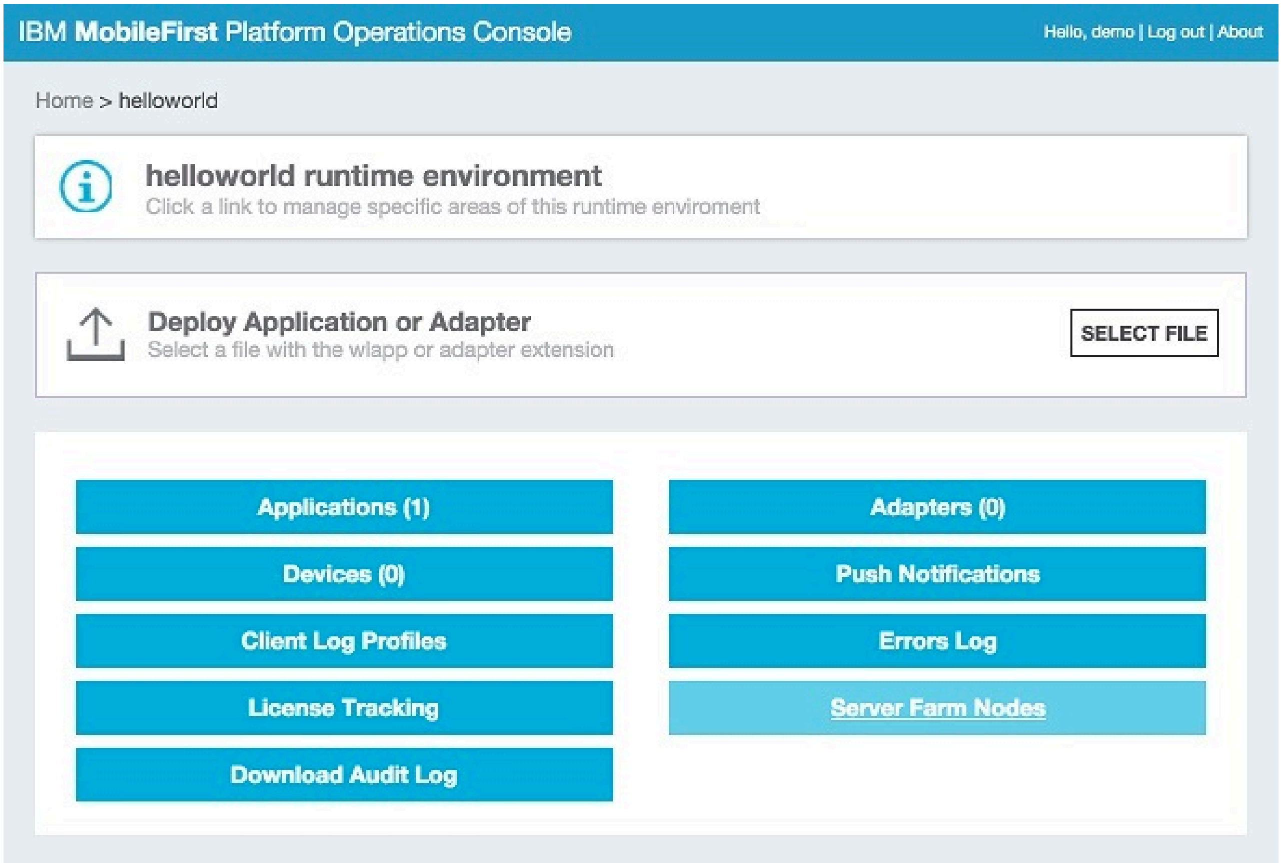
Configuring the heartbeat rate and timeout values

You can configure the heartbeat rate and timeout values by defining the following JNDI properties:

- **`ibm.worklight.admin.farm.heartbeat`**
- **`ibm.worklight.admin.farm.missed.heartbeats.timeout`**

For more information, see "List of JNDI properties for MobileFirst Server administration" on page 6-101

You can check the status of farm members from MobileFirst Operations Console by clicking **Server Farm Nodes**.



helloworld runtime environment

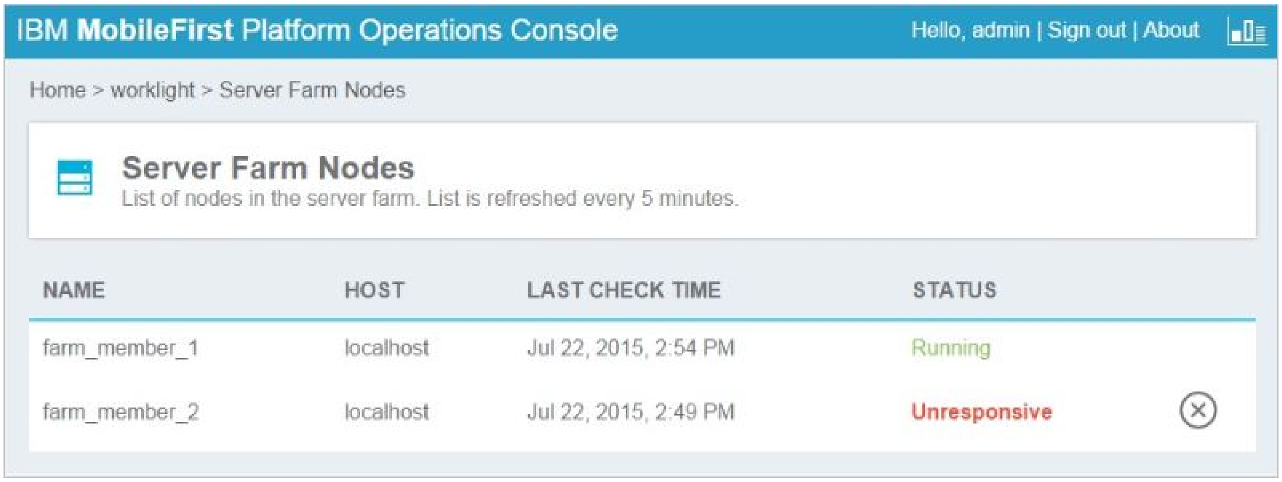
Click a link to manage specific areas of this runtime enviroment

Deploy Application or Adapter

Select a file with the wlapp or adapter extension

Figure 6-9. Checking the status of farm nodes

Clicking **Server Farm Nodes** enables you to access the list of registered farm members and their status.


IBM MobileFirst Platform Operations Console
Hello, admin | Sign out | About
Home > worklight > Server Farm Nodes

Server Farm Nodes

List of nodes in the server farm. List is refreshed every 5 minutes.

NAME	HOST	LAST CHECK TIME	STATUS
farm_member_1	localhost	Jul 22, 2015, 2:54 PM	Running
farm_member_2	localhost	Jul 22, 2015, 2:49 PM	Unresponsive ⊗

Figure 6-10. List of server farm nodes

The node identified as “farm_member_2” is considered to be down, which indicates that this server has probably failed and requires some maintenance.

Configuring MobileFirst Server

Consider your backup and recovery policy, optimize your MobileFirst 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 MobileFirst instance, for example in a source control system.

It is advisable to back up the runtime 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.

Disaster recovery: active-active topology

How to enable disaster recovery and high availability with IBM MobileFirst Platform Foundation for iOS by deploying an active-active topology.

Overview

When a mobile application is widely used around the world, in many cases the application servers are located in several data centers.

Starting from V7.1.0 of IBM MobileFirst Platform Foundation for iOS, if your system is operating in session-independent mode, you can provide high availability and disaster recovery to your distributed network. One possible topology to achive this is active-active. The topology involves several instances of IBM MobileFirst Platform Server that are deployed in two or more active data centers, in combinations with IBM Cloudant and IBM WebSphere eXtreme scale. Optionally, these configuration might use a load-balancer to determine which site handles the client requests.

Session-independence

A prerequisite for enabling an active-active topology is to have IBM MobileFirst Platform Server operating in session-independent mode.

For more information, see “Session-independent mode” on page 8-108.

Topologies for achieving high availability

Topologies that deliver active-active features must work with the same IBM MobileFirst Platform Foundation for iOS attribute store. Starting with V7.1.0, the attribute store in IBM MobileFirst Platform Server persists, by default, to the runtime database. There are two supported persistency configurations:

Cloudant

In this configuration, there is one administration database per data center. The runtime database is shared between both data centers on Cloudant, which provides high availability support. See Figure 1.

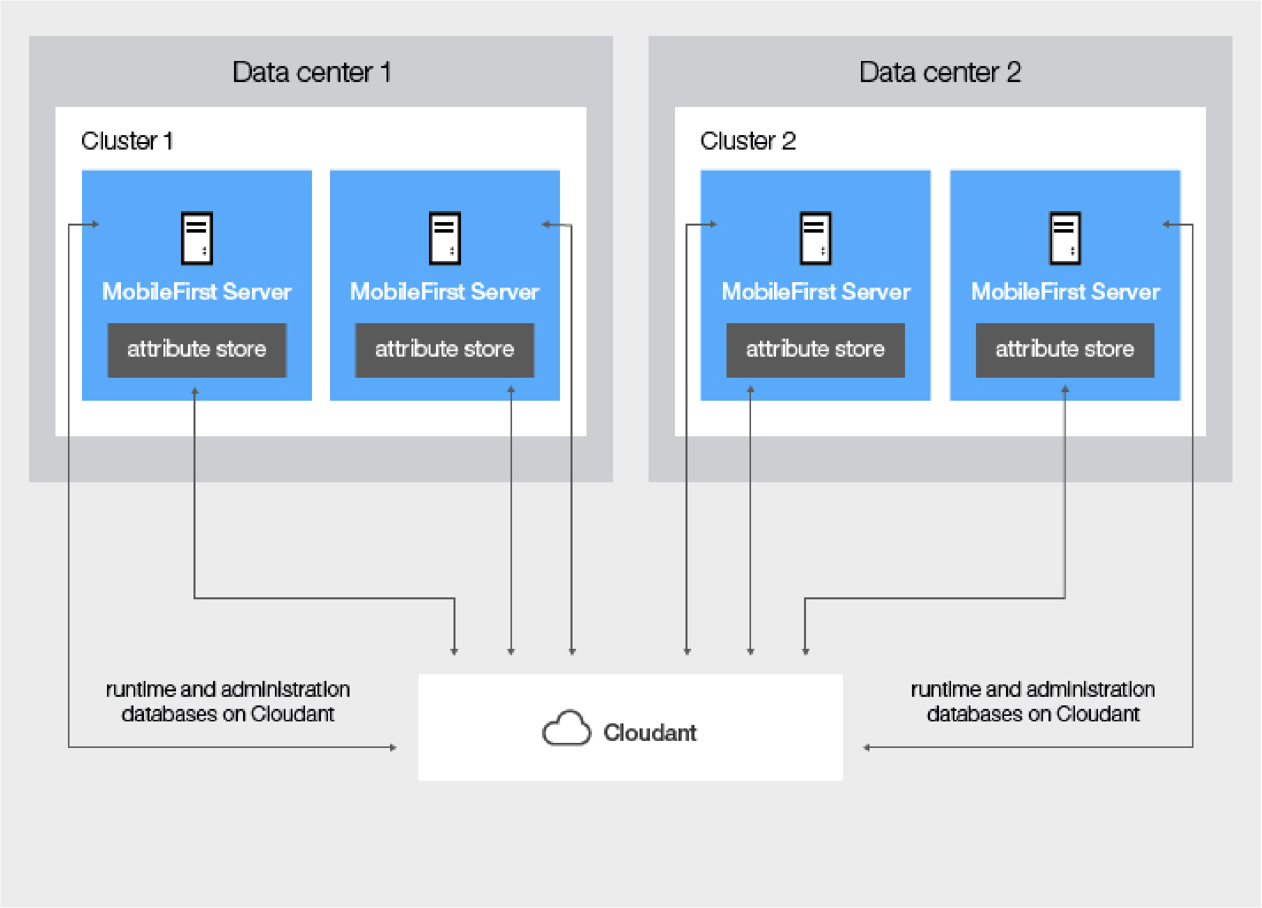


Figure 6-11. Active-active topology using Cloudant

Important:

- Ensure that runtime servers in all data centers are using the same Cloudant database.
- Because the administration console and database are not shared between data centers, MobileFirst artifacts such as applications and adapters must be deployed in each data center.

IBM WebSphere eXtreme Scale, persisted to Cloudant

In this configuration, eXtreme Scale provides an additional caching layer for increased performance. The layer is used to cache the attribute store, which is the most extensively used object in MobileFirst client login flows.The usage of eXtreme Scale alone does not provide built-in active-active functionality, as eXtreme Scale writes nothing to disk. All data in eXtreme Scale is managed in-memory. There are several ways to achieve active-active using eXtreme Scale. This page describes the configuration that involves eXtreme Scale and a built-in write-behind Loader plug-in that is provided with MobileFirst Server. The Loader persists the data in eXtreme Scale to a Cloudant database, as illustrated in Figure 2. When the attribute store data is persisted, even if one data center crashes along with its eXtreme Scale server(s), the eXtreme Scale server(s) in other data centers can restore their lost state from Cloudant and keep serving the mobile clients transparently.

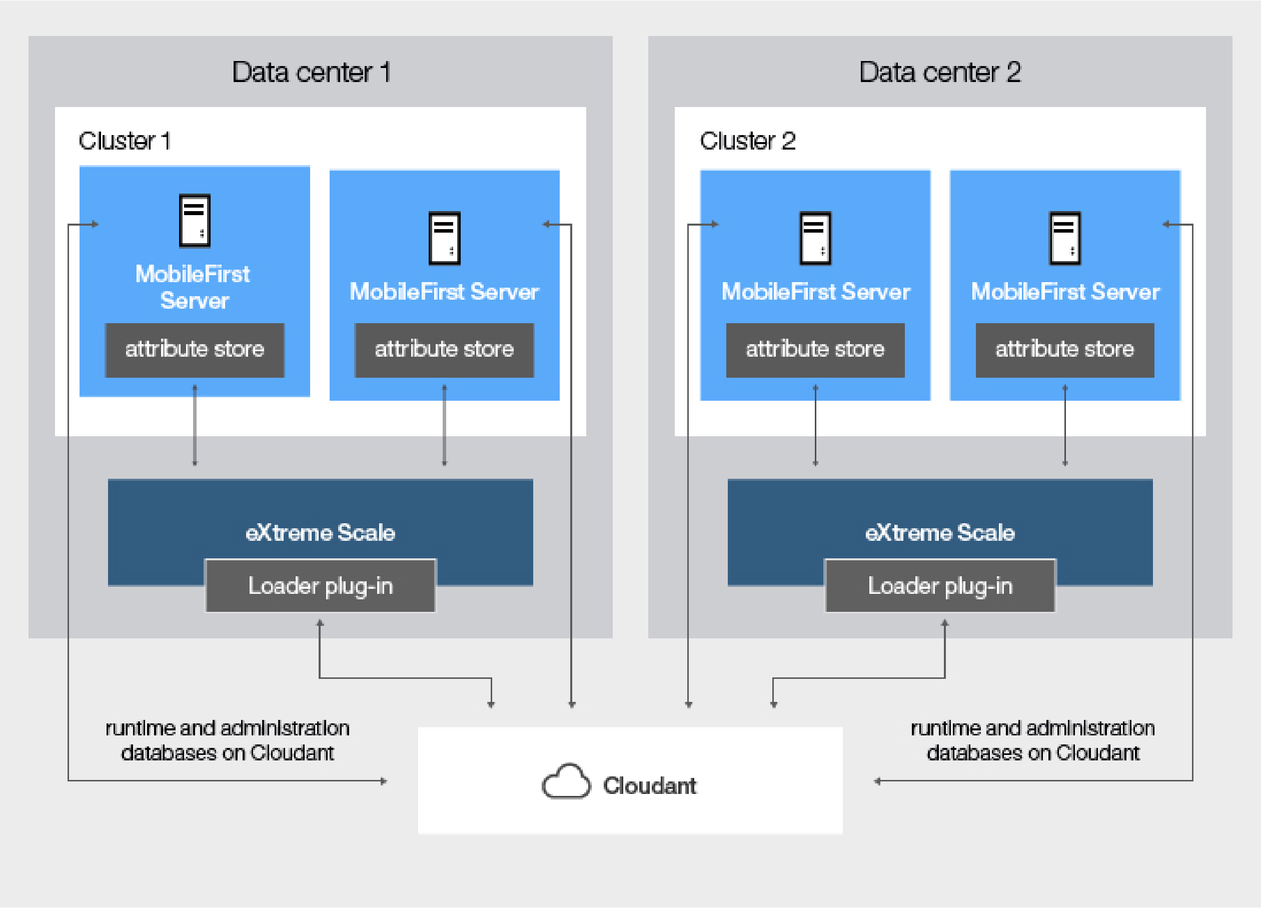


Figure 6-12. Active-active topology using eXtreme Scale and Cloudant

Configuring two data centers to work on Cloudant

There are two ways to configure two data centers to work in an active-active topology: with the Server Configuration Tool, or with an Ant task.

Server configuration tool

For each data center, run the Server Configuration Tool on all machines using the same Cloudant account.

1. For the administration services database, specify a name that is unique for the data center. For example, `admin_db_datacenter1` for the first data center and `admin_db_datacenter2` for the second data center.
2. For the runtime, specify the same prefix for the database name in all data centers.

Ant task

The relevant sample Ant file is located at `<`*`installDir`*`>/WorklightServer/configuration-samples`. You must specify values for the following properties:

- worklight.contextroot
- database.cloudant.url
- database.cloudant.username
- database.cloudant.worklight.dbprefix
- database.cloudant.wladmin.dbname

Specify the same value for the first four properties in all data centers. For the last property, specify a unique value for each data center. For example, `admin_db_datacenter1` and `admin_db_datacenter2`.

Configuring IBM MobileFirst Platform Foundation for iOS to work with eXtreme Scale and Cloudant

For more information, see “Attribute store over IBM WebSphere eXtreme Scale” on page 8-112.

Optimization and tuning of MobileFirst Server

Optimize the MobileFirst Server configuration by tuning the allocation of Java virtual machine (JVM) memory, HTTP connections, back-end connections, and internal settings.

The MobileFirst Server works with three application servers: Apache Tomcat, WebSphere Application Server and Liberty profile. For best results, install MobileFirst Server on a 64-bit operating system, and use only 64-bit software.

JDK

The MobileFirst Server can run on IBM JDK or Oracle JDK.

JVM memory allocation

The Java instance of the application server allocates memory. Consider the following general guidelines for JVM memory allocations:

- Set the JVM memory to at least 2 GB. This means you can not use less than 2GB, but that might not be enough and you will have to specify more, based on the requirements.
- For a production environment, setting the minimum heap size and maximum heap size to the same value can provide the best performance, as it avoids heap expansion and contraction.
- Set the required memory size of the application server:
  - `Liberty`: See the `jvm.options` section in Customizing the Liberty profile environment. 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.

For information about how to calculate memory size, see the Scalability and Hardware Sizing document and the hardware calculator spreadsheet at the Developer Center website for IBM MobileFirst Platform Foundation.

Tuning HTTP connections

This configuration defines threading and execution settings for the application server.

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

Specific application server configuration:

- `Liberty`: See the executor section in Liberty profile: Configuration elements in the server.xml file.

  By default, the maximum number of threads is unlimited.
- `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 HTTP Connector page in 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 database connections."
- If you expect a high number of concurrent users, increase the number of default threads.
- Liberty specific: Even though the maximum number of threads is unlimited, the executor service makes informed choices whether adding another thread will actually be useful.

Tuning database connections

In tuning database connections, the most important parameter is the number of connection threads from the server to the database. This configuration is made in the data source. This is mostly important when working in session-independent mode, where the attribute store is kept in the runtime database (that is, `mfp.attrStore.type` has been set to `true`). In this mode, the server makes extensive use of the database. In addition, there are two IBM MobileFirst Platform Foundation for iOS features that rely heavily on the database: SSO (single sign-on) and reports. When using these features, you must ensure that you have enough database connection threads. The only limitation is that each node in the MobileFirst Server cluster can have no more than **`MAX_DB_INCOMING_CONNECTIONS/NUM_OF_CLUSTER_NODES`** connection threads, where **`MAX_DB_INCOMING_CONNECTIONS`** is the maximum incoming connections defined in the database server and **`NUM_OF_CLUSTER_NODES`** is the number of MobileFirst Server nodes in the cluster. A rough rule of thumb is to set the number of database connections to be the number of HTTP threads in the application server, as long as you maintain the limitation above.

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.

For data source configuration, check the following topics:

- `WebSphere Application Server`: See Connection pool settings.
- `Apache Tomcat`: See JNDI Datasource HOW-TO.
- `Liberty Server`: See the Datasource section in Liberty profile: Configuration elements in the server.xml file.
- `Cloudant`: See Property keys and values for Cloudant configuration.

Tuning back-end connections

`maxConcurrentConnectionsPerNode`

> The **`maxConcurrentConnectionsPerNode`** parameter defines the maximum number of concurrent calls to the back-end service from the MobileFirst Server node. This **`maxConcurrentConnectionsPerNode`** parameter is set in the **`<connectionPolicy>`** element of the adapter XML file.
>
> Starting from IBM MobileFirst Platform Foundation for iOS V6.3, all requests to the back-end remain on the HTTP thread. The MobileFirst Server does NOT allocate a new thread for the backend request. The only use of **`maxConcurrentConnectionsPerNode`** is for blocking the number of connections to the HTTP back-end. The implication is that you can specify a large value for **`maxConcurrentConnectionsPerNode`** (for example, 5000), so as not to limit the back-end calls.

Handling slow backend servers

> If your backend server is slow, increase the values for your server settings, in particular the following values:
>
> - Number of HTTP threads in the application server: For a backend that responds in 750 ms, for example, 3000 HTTP threads is recommended.
> - **`maxConcurrentConnectionsPerNode`** in the adapter XML file: For a backend that responds in 750 ms, for example, 3000 is recommended.
> - OS settings: Increase the number of open files. 4096 is the recommended number.
> - Clients threads: A good rule of thumb is 2900 JMETER clients threads.
> - Backend server: 3000 threads is recommended.

Push Notifications

For push notification information see the Push Notification section in the Scalability and Hardware Sizing document and the hardware calculator spreadsheet at the Developer Center website for IBM MobileFirst Platform Foundation.

Analytics

For Analytics Server configuration see the Analytics section in the Scalability and Hardware Sizing document and the hardware calculator spreadsheet at the Developer Center website for IBM MobileFirst Platform Foundation.

MobileFirst Server internal configuration

When working in “Session-dependent mode” on page 8-111, consider the following factors:

- The **`serverSessionTimeout`** property defines client inactivity timeout, after which the session is invalidated. A session is an object stored in the server memory for each connecting device. Among other data, it stores authentication information. Active sessions are determined by the number of sessions opened versus the sessions timing out due to lack of activity. The default session timeout is 10 minutes, but it can and should be configured. Users typically set the timeout to anywhere from 5 to 10 minutes. This parameter affects the server memory consumption. Note that in session-independent mode, no data is stored in the session, therefore the session timeout is irrelevant.

- In addition, the mobile client has a "heartbeat" property that allows the mobile client to ping the server while the app is in the foreground, so that the server session will not time out.

  Note:
  When a mobile app has moved into the background, it no longer interacts with the server, nor sends a "heartbeat". The result is that the server session drops after the specified server session timeout.
- 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 will remain active on the server for 10 minutes, leaving 10 x 1,000 = 10,000 sessions.

Intervals for background tasks

The following **worklight.properties** parameters control the intervals at which background tasks. Background tasks perform several actions on the database and/or file system:

sso.cleanup.taskFrequencyInSeconds
The SSO (single sign-on) mechanism stores session data in a database table. This parameter is the interval for the SSO cleanup task to check if there are inactive accounts in the SSO table. If any are found, it deletes them. The default value is 5 seconds, meaning that every 5 seconds, the database is checked for inactive accounts. An inactive account is one that has remained idle for longer than the value of the **serverSessionTimeout** property.

push.cleanup.taskFrequencyInSeconds
Deletes inactive push notification subscriptions. The default is 60 minutes. This parameter is currently implemented only for Apple APNS.

mfp.attrStore.db.cleanupFrequency.minutes
Deletes stale attributes from the attribute store database table. The default is 60 minutes.

Optimization of MobileFirst Server project databases

You can improve the performance of the project databases or schemas that support MobileFirst Server.

Note: The Reports database and the associated APP_ACTIVITY_REPORT table described below are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

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

Database disks

You can find some overview information about the MobileFirst Server project databases in the **Database usage and size** section of the Scalability and Hardware Sizing document and its accompanying hardware calculator spreadsheet at the Developer Center website for IBM MobileFirst Platform Foundation. The spreadsheet 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 MobileFirst 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 MobileFirst Platform Foundation for iOS.

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 MobileFirst 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 MobileFirst Server performance

You can run performance tests on the different features of the MobileFirst Server. This section describes how to run the Apache jMeter performance test tool, but the procedure is similar for other tools.

Note: The procedures described in this page apply only to a scenario that involves a propriety JavaScript adapter that is running in **session-dependent** mode. It is not applicable in a scenario that involves the OAuth security framework. For more information about session independency, see "Session-independent mode" on page 8-108. For more information about OAuth, see "OAuth-based security model" on page 8-188.

The following features can have an impact on MobileFirst 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 MobileFirst Server performance.

Testing authentication flow performance

The following realm, which is part of the default security test for iOS, is 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.

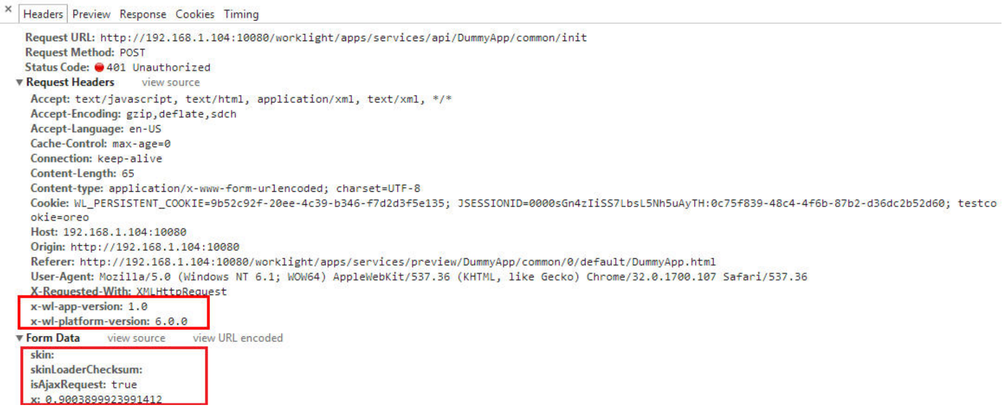
For more information about the realms, see "The authentication configuration file" on page 8-253.

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 MobileFirst 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 6-33. Initialization parameters

Parameter	Description
`x-wl-app-version`	Application version.
`x-wl-platform-version`	Version of the product that built the application.

This is an example of a jMeter test:

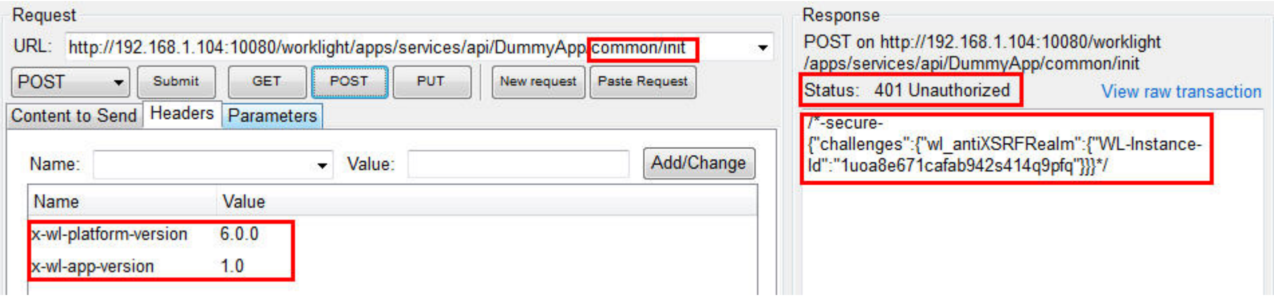


```
Request URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/common/init
Request Method: POST
Status Code: ● 401 Unauthorized
▼ Request Headers    view source
  Accept: text/javascript, text/html, application/xml, text/xml, */*
  Accept-Encoding: gzip,deflate,sdch
  Accept-Language: en-US
  Cache-Control: max-age=0
  Connection: keep-alive
  Content-Length: 65
  Content-type: application/x-www-form-urlencoded; charset=UTF-8
  Cookie: WL_PERSISTENT_COOKIE=9b52c92f-20ee-4c39-b346-f7d2d3f5e135; JSESSIONID=0000sGn4zIiSS7LbsL5Nh5uAyTH:0c75f839-48c4-4f6b-87b2-d36dc2b52d60; testco
  okie=oreo
  Host: 192.168.1.104:10080
  Origin: http://192.168.1.104:10080
  Referer: http://192.168.1.104:10080/worklight/apps/services/preview/DummyApp/common/0/default/DummyApp.html
  User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.107 Safari/537.36
  X-Requested-With: XMLHttpRequest
  x-wl-app-version: 1.0
  x-wl-platform-version: 6.0.0
▼ Form Data    view source    view URL encoded
  skin:
  skinLoaderChecksum:
  isAjaxRequest: true
  x: 0.9003899923991412
```

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 MobileFirst initialization service

The response data from the MobileFirst `init` request differs depending on the security test you apply on your MobileFirst 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.



Request

URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/common/init

Name:

Value:

Name	Value
x-wl-platform-version	6.0.0
x-wl-app-version	1.0

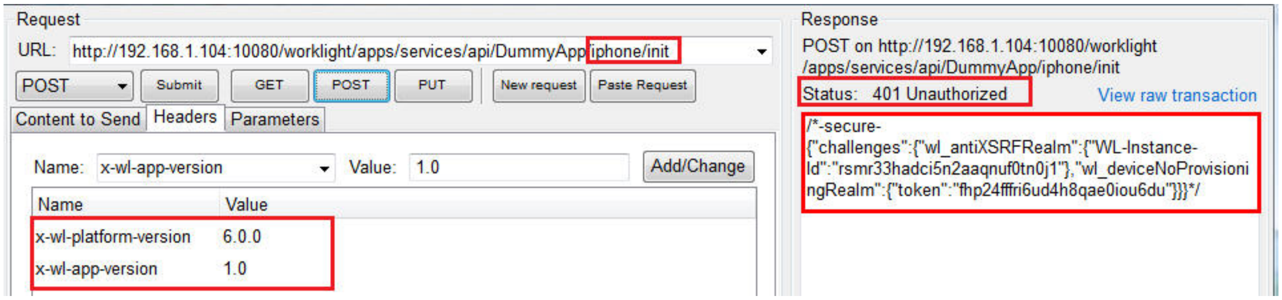
Response

POST on http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/common/init

Status: 401 Unauthorized

```
/*-secure-
{"challenges":{"wl_antiXSRFRealm":{"WL-Instance-Id":"1uoa8e671cafab942s414q9pfq"}}}*/
```

Figure 6-13. Response data from common environment



URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/iphone/init

Name: x-wl-app-version Value: 1.0

Name	Value
x-wl-platform-version	6.0.0
x-wl-app-version	1.0

POST on http://192.168.1.104:10080/worklight
/apps/services/api/DummyApp/iphone/init

Status: 401 Unauthorized

```
/*-secure-
{"challenges":{"wl_antiXSRFRealm":{"WL-Instance-
Id":"rsmr33hadci5n2aaqnuf0tn0j1"},"wl_deviceNoProvisioni
ngRealm":{"token":"fhp24fffri6ud4h8qae0iou6du"}}}*/
```

Figure 6-14. 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 MobileFirst 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" later in this section.

Changing the response status to HTTP 200

When the performance testing thread runs the initialization for the first time, MobileFirst 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.

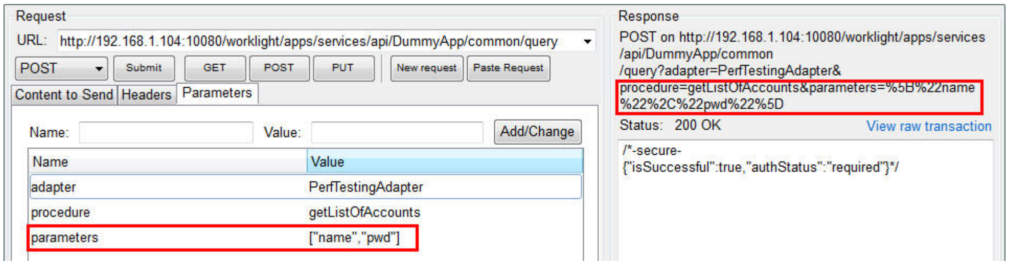
Testing back-end invocation

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 6-34. Backend invocation parameters

Parameter	Description
`adapter`	MobileFirst Adapter name.
`procedure`	MobileFirst procedure name
`parameters`	Procedure parameters should be an array.

The following figure shows an example array of parameters:



URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/common/query

Name: Value:

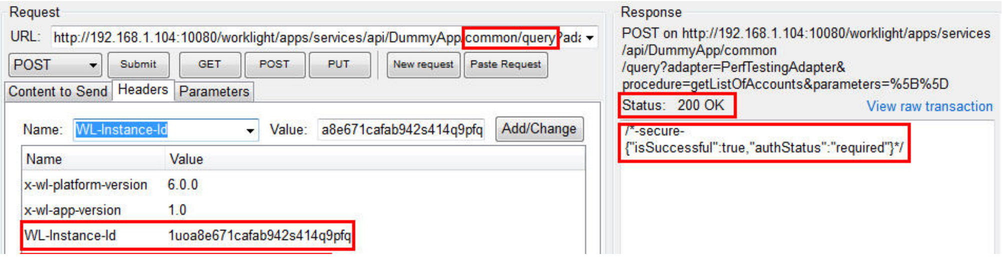
Name	Value
adapter	PerfTestingAdapter
procedure	getListOfAccounts
parameters	["name","pwd"]

POST on http://192.168.1.104:10080/worklight/apps/services /api/DummyApp/common /query?adapter=PerfTestingAdapter& procedure=getListOfAccounts&parameters=%5B%22name %22%2C%22pwd%22%5D

Status: 200 OK

```
/*-secure-
{"isSuccessful":true,"authStatus":"required"}*/
```

The following figure shows an example of request headers:



URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/common/query?ad

Name: WL-Instance-Id Value: a8e671cafab942s414q9pfq

Name	Value
x-wl-platform-version	6.0.0
x-wl-app-version	1.0
WL-Instance-Id	1uoa8e671cafab942s414q9pfq

POST on http://192.168.1.104:10080/worklight/apps/services /api/DummyApp/common /query?adapter=PerfTestingAdapter& procedure=getListOfAccounts&parameters=%5B%5D

Status: 200 OK

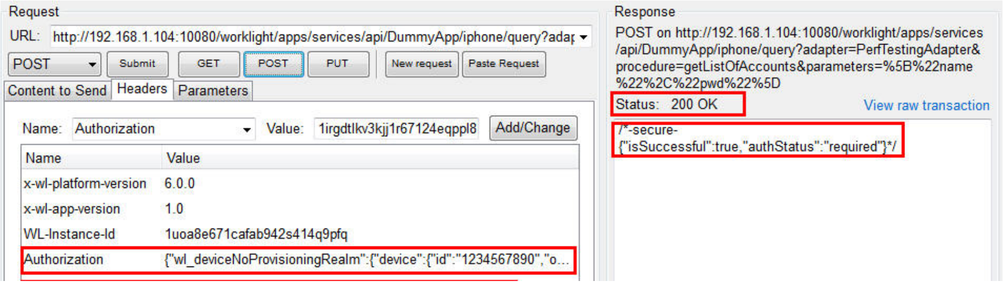
```
/*-secure-
{"isSuccessful":true,"authStatus":"required"}*/
```

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 environment, since it contains `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 MobileFirst Adapter procedure was successfully received and now you can continue with your back-end testing.



URL: http://192.168.1.104:10080/worklight/apps/services/api/DummyApp/iphone/query?adap

Name: Authorization Value: 1irgdtlkv3kjj1r67124eqppl8

Name	Value
x-wl-platform-version	6.0.0
x-wl-app-version	1.0
WL-Instance-Id	1uoa8e671cafab942s414q9pfq
Authorization	{"wl_deviceNoProvisioningRealm":{"device":{"id":"1234567890","o...

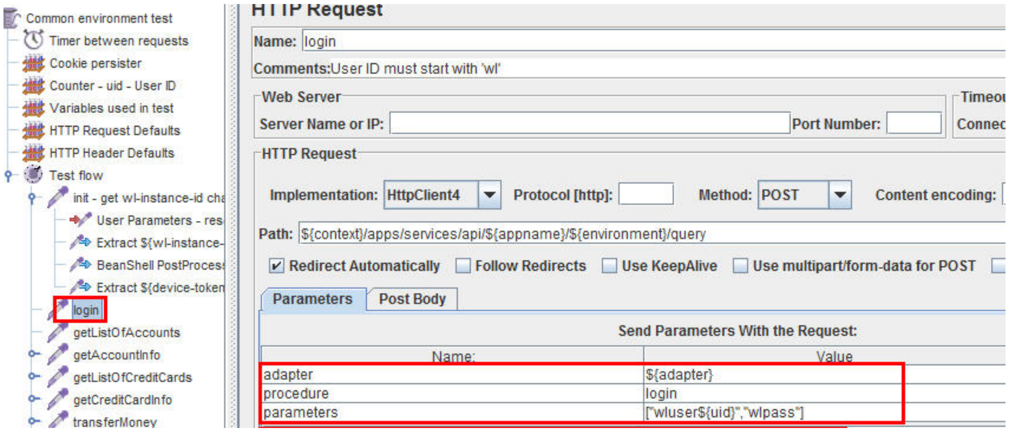
POST on http://192.168.1.104:10080/worklight/apps/services /api/DummyApp/iphone/query?adapter=PerfTestingAdapter& procedure=getListOfAccounts&parameters=%5B%22name %22%2C%22pwd%22%5D

Status: 200 OK

```
/*-secure-
{"isSuccessful":true,"authStatus":"required"}*/
```

Logging in

When the MobileFirst adapter procedure is protected by a security test or the `connectAs` property is set to `endUser`, you need to log in to the MobileFirst Server before calling this procedure. To check if the MobileFirst adapter procedure needs a login, you can call the procedure followed by the steps described earlier, and check the response data from the MobileFirst Server. If the response data includes `isSuccessful:true` and `authStatus:required`, you should log in to the MobileFirst Server first, otherwise the requests to this procedure are rejected by MobileFirst Server. The way you log in to the MobileFirst 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 MobileFirst 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 MobileFirst Server. The following screen image shows an example of a login function that calls the `setActiveUser` API with the supplied user ID and password:



HTTP Request

Name: login

Comments:User ID must start with 'wl'

Server Name or IP:

Port Number:

HTTP Request

Implementation: HttpClient4

Protocol [http]:

Method: POST

Content encoding:

Path: ${context}/apps/services/api/${appname}/${environment}/query

☑ Redirect Automatically ☐ Follow Redirects ☐ Use KeepAlive ☐ Use multipart/form-data for POST

Send Parameters With the Request:

Name:	Value
adapter	${adapter}
procedure	login
parameters	["wluser${uid}","wlpass"]

Logging out

The following options are available for logging out of the MobileFirst Server:

Not logging out for each iteration

MobileFirst 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 MobileFirst 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 6-336.

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 “(deprecated) Reports database” on page 13-94. After you enable database reporting, you can use the back-end invocation step described earlier. See the database reporting section in the Scalability and Hardware Sizing document at the Developer Center website for IBM MobileFirst Platform Foundation.

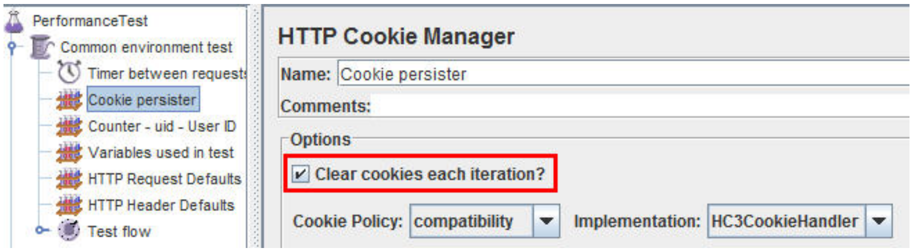
Single sign-on (SSO), direct update, push notification, and geolocation

See the relevant section in the Scalability and Hardware Sizing document at the Developer Center website for IBM MobileFirst Platform Foundation.

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 Cookie Manager

Name: Cookie persister

Comments:

Options

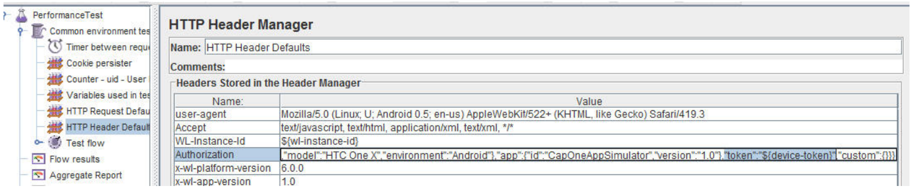
☑ Clear cookies each iteration?

Cookie Policy: compatibility

Implementation: HC3CookieHandler

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:



HTTP Header Manager

Name: HTTP Header Defaults

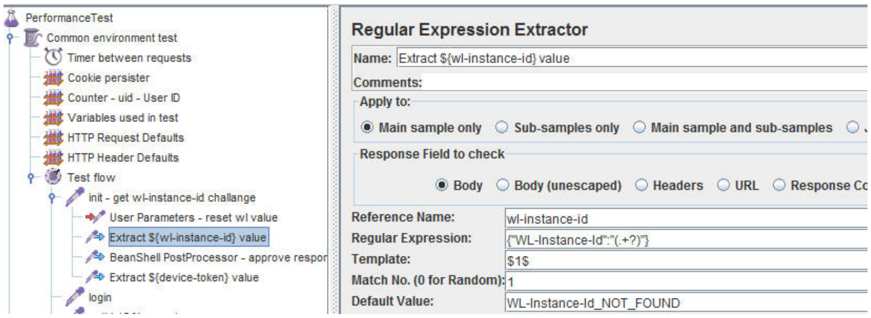
Comments:

Headers Stored in the Header Manager

Name:	Value
user-agent	Mozilla/5.0 (Linux; U; Android 0.5; en-us) AppleWebKit/522+ (KHTML, like Gecko) Safari/419.3
Accept	text/javascript, text/html, application/xml, text/xml, */*
WL-Instance-Id	${wl-instance-id}
Authorization	,"model":"HTC One X","environment":"Android"},"app":{"id":"CapOneAppSimulator","version":"1.0"},"token":"${device-token}","custom":{}}}
x-wl-platform-version	6.0.0
x-wl-app-version	1.0

Initialization phase

1. Extract and replace the `WL-Instance-Id` placeholder:



Regular Expression Extractor

Name: Extract ${wl-instance-id} value

Comments:

Apply to:

● Main sample only ○ Sub-samples only ○ Main sample and sub-samples ○

Response Field to check

● Body ○ Body (unescaped) ○ Headers ○ URL ○ Response Co

Reference Name: wl-instance-id

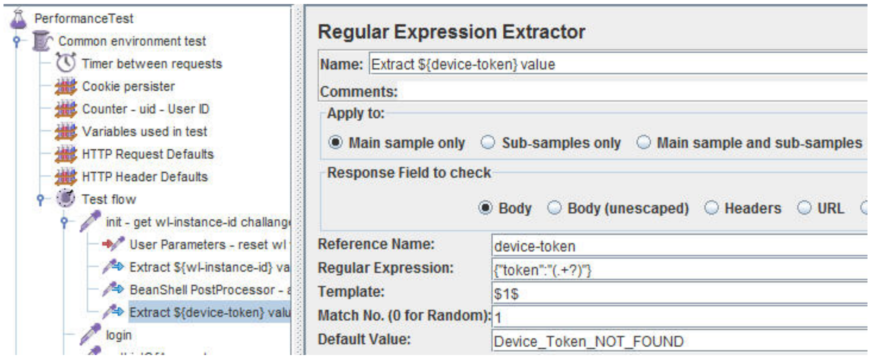
Regular Expression: {"WL-Instance-Id":"(.+?)"}

Template: 1

Match No. (0 for Random): 1

Default Value: WL-Instance-Id_NOT_FOUND

2. Extract and replace the `WL_deviceNoProvisioningRealm` token placeholder:



Regular Expression Extractor

Name: Extract ${device-token} value

Comments:

Apply to:

● Main sample only ○ Sub-samples only ○ Main sample and sub-samples

Response Field to check

● Body ○ Body (unescaped) ○ Headers ○ URL

Reference Name: device-token

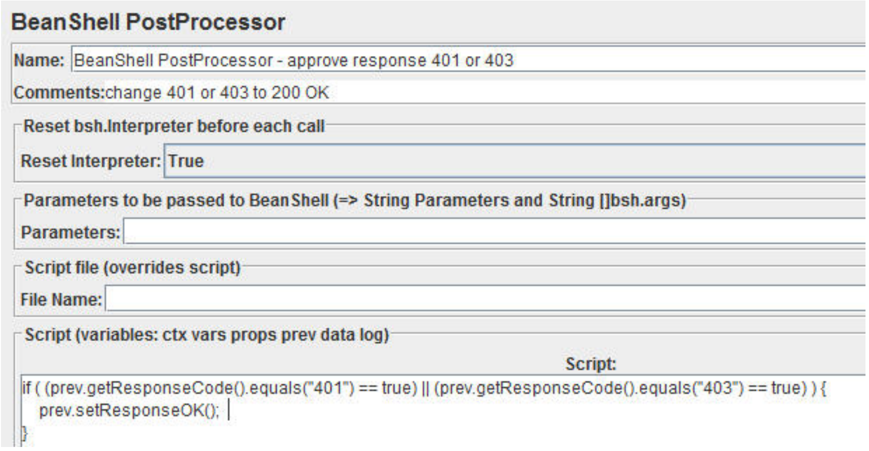
Regular Expression: {"token":"(.+?)"}

Template: 1

Match No. (0 for Random): 1

Default Value: Device_Token_NOT_FOUND

3. Change initialization HTTP status 401 and 403 to HTTP status 200:



BeanShell PostProcessor

Name: BeanShell PostProcessor - approve response 401 or 403

Comments:change 401 or 403 to 200 OK

Reset bsh.Interpreter before each call

Reset Interpreter: True

Parameters to be passed to BeanShell (=> String Parameters and String []bsh.args)

Parameters:

Script file (overrides script)

File Name:

Script (variables: ctx vars props prev data log)

Script:

```
if ( (prev.getResponseCode().equals("401") == true) || (prev.getResponseCode().equals("403") == true) ) {
   prev.setResponseOK();
}
```

Security configuration

Configure the security of the MobileFirst Server as detailed here.

Securing the MobileFirst Server administration

No unauthorized person can perform MobileFirst 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 can modify the behavior of these apps on the mobile devices. The apps are served to the clients through the MobileFirst 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 IBM MobileFirst Platform Operations 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 either <feature>ssl-1.0</feature> or <feature>restConnector-1.0</feature>, or both features. The restConnector-1.0 feature implies that the ssl-1.0 feature is enabled.

- Verify that the HTTPS port is enabled, by ensuring that the `server.xml` file does not have an `<httpEndpoint>` element with a negative `httpsPort` attribute. If the HTTPS port is disabled, SSL is also disabled, and the JMX connections that the MobileFirst 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 MobileFirst 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 MobileFirst administration operations are possible. Here are steps to mitigate this risk.

- Minimize the number of users that you map to the `worklightadmin` and `worklightdeployer` roles. For more information, see "Configuring user authentication for MobileFirst Server administration" on page 6-96.
- Map different users to the `worklightadmin` or `worklightdeployer` roles 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"`), mapping users and roles differently 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 MobileFirst Operations Console or the MobileFirst Administration REST services over `http`. Always use `https`. You can prevent such use in the following ways:
  - 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 Java EE 6 transport security of type `CONFIDENTIAL`. This setting redirects 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 MobileFirst Operations Console and Administration Services with Ant tasks" on page 6-64.

- Never use the <wladm> Ant task with the attribute secure="false", and never use the **wladm** command with the option -secure=false. For this purpose:
  - Ensure that your application server uses an SSL certificate signed by a CA, not a self-signed certificate, and that the host name that is mentioned in this certificate matches the host name of the application server host.
  - 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 so, 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 MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst Operations 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 computers that serve this database.
- If you use Ant tasks to configure the MobileFirst Server administration (see "Using Ant tasks to install MobileFirst Server administration" on page 6-62), 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 15-66.

  - 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 computers that run MobileFirst 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 15-66.

Protecting the JMX communication

If the JMX communication between Administration Services and the MobileFirst runtime environments are not secured, unauthorized persons who have local access to the MobileFirst Server computers 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, 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 6-54.

For WebSphere Application Server Liberty, encode the password of the user who is granted the administrator role that is required for the JMX communication. This password is declared in the Liberty profile `server.xml` file, both in the **`ibm.worklight.admin.jmx.pwd`** JNDI property and inside the `<user>` element of the basic registry in case you use this registry type. Both passwords can be encoded with the Liberty profile `securityUtility` program. The supported encoding types are `xor` and `aes` (only with the default key).

Protecting the apps and adapters to deploy

If the source from which the MobileFirst administrator receives apps and adapters is not secured, tampered apps and adapters can be submitted to the MobileFirst administrator, who then deploys them. Here are steps to mitigate this risk.

- Ensure that the MobileFirst administrator receives apps and adapters only through channels that 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 version control systems in this category are RTC/jazz and Git. In contrast, CVS does not belong to this category.

Protecting against attacks from the internet

Attackers from the internet might attempt to search for security flaws in the MobileFirst Operations 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 MobileFirst Server from the internet, but all legitimate uses of the MobileFirst Operations 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 MobileFirst Operations

Console (default: /worklightconsole) and of the Administration Services (default: /worklightadmin). At the same time, keep the access to the MobileFirst runtime web applications open.

Database and certificate security passwords

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

Configuration of a IBM MobileFirst Platform Server typically includes the following credentials:

- User name and password to the runtime 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 MobileFirst applications on the server" on page 11-50 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 11-61.

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 MobileFirst Server in WebSphere Application Server with Java 2 security enabled

You can run IBM MobileFirst Platform 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 MobileFirst 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.

Integration with IBM WebSphere DataPower as a security gateway and reverse proxy

Protect your mobile-application traffic by using IBM WebSphere DataPower as a reverse proxy and security gateway in the DMZ between client applications and MobileFirst Server.

Protecting mobile-application traffic that enter your network from customer and employee devices involves preventing data from being altered, authenticating users, and allowing only authorized users to access applications. You can use the DataPower security-gateway features of to protect mobile-application traffic that is initiated by client MobileFirst applications.

Enterprise topologies are designed to include different protection zones so that specific processes can be secured and optimized. You can use DataPower in different ways in the DMZ (a firewall configuration for securing local area networks) and in other zones within your network to protect enterprise resources. When you start to build MobileFirst applications to be delivered to the devices of your customers and employees, you can apply these methods to protect the mobile-application traffic.

You can use DataPower as a front-end reverse proxy and security gateway. DataPower uses a multiprotocol gateway (MPGW) service to proxy and secure access to MobileFirst mobile applications. You can select the method that DataPower will use to authenticate the mobile client, such as HTTP basic authentication or HTML forms-based authentication. The following topics demonstrate how to implement this topology by using either HTTP basic authentication or HTML forms-based authentication. You can adjust the procedure, as needed, to use a different authentication method. For more information about configuring DataPower, see the WebSphere DataPower documentation.

Consider adopting the following phased approach to establishing DataPower as a reverse proxy and security gateway:

1. Install and configure a MobileFirst environment, and test the installation with a simple application without DataPower acting as the reverse proxy.
2. Test your application logic and verify that it works.
3. Configure your MobileFirst project to work with your preferred reverse-proxy DataPower gateway configuration.
4. Configure a multi-protocol gateway on your DataPower appliance to use DataPower as a proxy for your MobileFirst mobile application or MobileFirst Operations Console. As part of the configuration select your preferred authentication method for the DataPower AAA (authentication, authorization, audit) policy.
5. Run your application and attempt to access a protected resource to test the implementation.

Note: To integrate your application with DataPower by using LTPA and DataPower HTML forms-based authentication, follow the specialized procedure “Integrating with DataPower as a reverse proxy using LTPA and form-based authentication” on page 6-168. This procedure includes a DataPower configuration pattern and MobileFirst samples that eliminate much of the manual configuration described in the more generic procedure “Integrating with DataPower as a security gateway and reverse proxy.”

Related information

- “Integration with IBM WebSphere DataPower” on page 14-11
- “Integrating IBM WebSphere DataPower with a cluster of instances of MobileFirst Server” on page 6-320
- “Integration and authentication with a reverse proxy” on page 14-3
- “Using a reverse proxy with server farm and WebSphere Application Server Network Deployment topologies” on page 6-18
- “Reverse proxy with LTPA” on page 11-97

Integrating with DataPower as a security gateway and reverse proxy

Learn how to integrate with IBM WebSphere DataPower as a reverse proxy and security gateway.

Before you begin

Note: To integrate your application with DataPower by using LTPA and DataPower HTML forms-based authentication, follow the specialized procedure at “Integrating with DataPower as a reverse proxy using LTPA and form-based authentication” on page 6-168. The specialized procedure includes a DataPower configuration pattern that eliminates much of the manual configuration described in the following generic procedure, including the need for the manual rule definitions outlined in the topic “DataPower configuration rules” on page 6-162.

- Read the topic “Integration with IBM WebSphere DataPower as a security gateway and reverse proxy” on page 6-156.
- Install IBM WebSphere DataPower by following the instructions in the WebSphere DataPower documentation.

- Install IBM MobileFirst Platform Foundation for iOS with interim fix 7.1.0.00-20160513-1006 or later.
  Ensure that the installation includes IBM MobileFirst Platform Command Line Interface for iOS.
- Establish a stand-alone MobileFirst Server environment that uses WebSphere Application Server.

Procedure

1. Configuring your MobileFirst applications
   a. For each application 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>
```

   By default, 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 MobileFirst Operations Console enterprise application.
2. Update your client mobile app.
   a. In your client mobile app, add the following JavaScript code to your HTML MobileFirst 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 MobileFirst 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.submitLoginFormCallback);

});
```

b. 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 substep 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, mp4, mp3"/>
  </security>
</ipad>
```

3. Define a multiprotocol gateway.
   a. In the IBM DataPower WebGUI, in the search box under Control Panel, enter `Multi-Protocol` and select **New Multi-Protocol Gateway**.
   b. On the General Configuration page, define the following settings:

Table 6-35. General Configuration

Field	**Description**
Multi-Protocol Gateway Name	Provide a name for your gateway.
Response Type	Select **Non-XML**. With this value, HTTP web application traffic (including JSON, JavaScript, and CSS) passes through the appliance.
Request Type	Select **Non-XML**. With this value, HTTP web application requests are handled by the appliance.
Front Side Protocol	Select **HTTPS (SSL)**. For this type of interaction in which user credentials are passed between client and server, HTTPS is appropriate. Also provide the following 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 6-37 on page 6-161. **Allowed Methods and Versions** Select **GET method** to enable support for HTTP Get. **SSL Proxy** Select an SSL Reverse Proxy profile to identify the SSL server.
Multi-Protocol Gateway Policy	Click **+**, and then create rules to define the policies that are listed in the following topics, depending on the type of authentication that you decide to use: • Policy **`worklight-basicauth`** for HTTP basic authentication. See "Rules for HTTP basic authentication" on page 6-162. • Policy **`mpgw-form`** for HTML forms-based login authentication. See "Rules for HTML forms-based authentication" on page 6-164.
Backend URL	Specify the address and port of the MobileFirst Server that is hosted on WebSphere Application Server.

4. Create an AAA policy that supports the HTTP basic authentication or HTML forms-based login policy that 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. Depending on the type of authentication that you want to use, define the following settings.
      - For HTTP basic authentication, specify the settings as listed in the following table.

Table 6-36. 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**.

      - For HTML forms-based authentication, specify the settings as listed in the following table.

Table 6-37. AAA policy for HTML forms-based authentication

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

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

Table 6-38. Advanced settings

Field	Value
Persistent Connections	**On**.
Allow Cache-Control Header	**Off**
Loop Detection	**Off**

Table 6-38. Advanced settings (continued)

Field	Value
Follow Redirects	**Off**. This value 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 MobileFirst application traffic is now protected by a secure DataPower gateway. User authentication is enforced by the DataPower appliance, which forwards the user credentials to MobileFirst Server either as an HTTP header or as an LTPA token, depending on the selected authentication method.

DataPower configuration rules:

Learn how to add the required authentication rules to your IBM WebSphere DataPower configuration.

Note: The following topics document how to perform specific types of manual DataPower configuration to complete the configuration steps outlined in the "Integrating with DataPower as a security gateway and reverse proxy" on page 6-157 procedure. If you use a preconfigured DataPower pattern, for example if you follow the specialized "Integrating with DataPower as a reverse proxy using LTPA and form-based authentication" on page 6-168 procedure, you do not need typically to perform these manual configurations.

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 "Integrating with DataPower as a security gateway and reverse proxy" on page 6-157, Table 6-35 on page 6-160.

Table 6-39. HTTP Basic Authentication properties

Property	Value
Policy Name	`worklight-basicauth`
Order of configured rules	1. `worklight-basicauth_rule_0`: see Table 6-40 on page 6-163 2. `worklight-basicauth_rule_3`: see Table 6-43 on page 6-163 3. `worklight-basicauth_rule_1`: see Table 6-41 on page 6-163 4. `worklight-basicauth_rule_2`: see Table 6-42 on page 6-163

Table 6-40. 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 6-41. 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 6-42. 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 6-167. • Output: NULL
Result	Not applicable.

Table 6-43. 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

Table 6-43. Properties of `worklight-basicauth_rule_3` (continued). 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
Result	Not applicable.

Rules for HTML forms-based authentication:

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

You create the `mpgw-form` policy as part of the process of defining a multi-protocol gateway. See "Integrating with DataPower as a security gateway and reverse proxy" on page 6-157, Table 6-35 on page 6-160.

Table 6-44. HTTP Form-Based Login properties

Property	Value
Policy Name	`mpgw-form`
Order of configured rules	1. `mpgw-form_rule_0`: see Table 6-45 2. `mpgw-form_rule_1`: see Table 6-46 3. `mpgw-form_rule_2`: see Table 6-47 on page 6-165 4. `mpgw-form_rule_3`: see Table 6-48 on page 6-165 5. `mpgw-form_rule_6`: see Table 6-49 on page 6-165

Table 6-45. 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 6-46. 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**.
Match	• Type = HTTP • HTTP header tag = Cookie • HTTP value match = `*LtpaToken*`
AAA	VerifyLTPA • Output: NULL
Result	Not applicable.

Table 6-47. 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(l)?(\?originalUrl=.*)?`
Transform	Provide a custom stylesheet that builds either a Login or Error HTML page. For a sample stylesheet, see "Sample form login stylesheet." **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 6-48. 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 6-49. 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 = *
Filter	Provide a custom stylesheet that handles redirect and content-type rewrite. For a sample redirect stylesheet, see "Sample redirect stylesheet" on page 6-167. - Output: NULL
Result	Not applicable.

Sample form login stylesheet:

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

You provide a custom stylesheet when defining rule `mpgw-form_rule_2`. See “Rules for HTML forms-based authentication” on page 6-164, Table 6-47 on page 6-165.

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

Sample redirect stylesheet:

You can use this sample stylesheet to handle redirection and content-type rewriting. You refer to the stylesheet when you create rules to define an HTTP basic authentication policy or an HTML forms-based authentication policy.

You provide a custom stylesheet when you define rule `mpgw-form_rule_6` (see "Rules for HTML forms-based authentication" on page 6-164, Table 6-49 on page 6-165), and when you define rule `worklight-basicauth_rule_2` (see "Rules for HTTP basic authentication" on page 6-162, Table 6-42 on page 6-163).

```
<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-response-
              <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>
```

Integrating with DataPower as a reverse proxy using LTPA and form-based authentication

Use the provided MobileFirst samples and IBM WebSphere DataPower configuration pattern and to integrate your application with a DataPower gateway that uses lightweight third-party authentication (LTPA) and HTML forms-based authentication.

Before you begin

- Read the topic “Integration with IBM WebSphere DataPower as a security gateway and reverse proxy” on page 6-156.
- Install IBM WebSphere DataPower by following the instructions in the WebSphere DataPower documentation.
  The following procedure and the sample DataPower configuration pattern (MFP_LTPA_Integration) were tested with DataPower Service Gateway XG45 version 7.0.0.10.
- Install IBM MobileFirst Platform Foundation for iOS with interim fix 7.1.0.00-20160513-1006 or later.
  Ensure that the installation includes IBM MobileFirst Platform Command Line Interface for iOS.
- Establish a stand-alone MobileFirst Server environment that uses WebSphere Application Server with LTPA.
- Download the MobileFirst LTPA DataPower integration package or the specific components that you require. The package contains the following artifacts:

- `pattern` contains a sample DataPower pattern (MFP_LTPA_Integration) that is preconfigured to integrate with IBM MobileFirst Platform Foundation for iOS using LTPA with HTML forms-based authentication. This pattern is used in the following procedure.
- `DataPower` contains a MobileFirst project with common platform-independent components.
- `DataPowerSwift` contains a sample Xcode project for a native iOS MobileFirst application.

About this task

Following is a specialized procedure for protecting mobile-application traffic by integrating your MobileFirst application with a DataPower gateway that uses LTPA and DataPower HTML forms-based login authentication. The procedure uses a sample MobileFirst DataPower configuration pattern (MFP_LTPA_Integration) that simplifies the required configuration of the DataPower gateway.

- Configuring your MobileFirst client application
- Configuring your MobileFirst Server
- Protecting your resources
- Configuring DataPower
- Customization notes
- Results and flow diagrams

Procedure

- **Configuring your MobileFirst client application**
  1. Create a MobileFirst application with your required application logic. See "Developing MobileFirst applications" on page 8-1. Start with a simple application that does not include security tests, and run the application to test your installation. You can also start out by using the sample project from the MobileFirst LTPA DataPower integration package.
  2. Add a challenge handler to your client application: When the application attempts to access a protected resource, the sample MFP_LTPA_Integration DataPower pattern sends a login forum to the application. Add a custom challenge handler to your application to detect the incoming login form by searching the response for a known string, such as `j_security_check`.

     The challenge-handler requirements for handling a DataPower login forum are the same as for handling MobileFirst form-based authentication. See "Implementing form-based authenticators" on page 8-274, Code the client side step. Note the following points:
     - The login forum's submit-action URL is `/j_security_check`.
     - The login forum that you submit back needs to contain `j_username` and `j_password` fields.

     The sample client application code in the MobileFirst LTPA DataPower integration package defines the required challenge handler for integrating with the sample DataPower MFP_LTPA_Integration pattern.
- **Configuring your MobileFirst Server**
  1. Configure the user registry in your application-server configuration file (`server.xml`):
     a. Open the LTPA keys file of your application server: `ltpa.keys`. (This file is typically found in the <*server_configuration_directory*>`/servers/worklight/resources/security` directory of your MobileFirst Server.) Find the `com.ibm.websphere.ltpa.Realm` key in the file, and copy and save its value (for example, `worklightRealm`).

b. Open the configuration file of your application server: server.xml. (This file is typically found in the <*server_configuration_directory*>/servers/worklight directory of your MobileFirst Server.) Look for a <basicRegistry> or <ldapRegistry> element in the file. If neither element exists, create the element that matches your required configuration (see the documentation of your application server).

c. Set the **realm** attribute of your user-registry element to the value of the com.ibm.websphere.ltpa.Realm LTPA key that you copied and saved in Step 1a on page 6-169 (if it is not already set).

d. When using the <basicRegistry> element, ensure that the element defines all the authorized users for your application. For example, the following basic-registry definition includes the credentials of two of the authorized users that are configured in the sample MFP_LTPA_Integration pattern:

```
<basicRegistry realm="worklightRealm">
    <user name="admin" password="admin"/>
    <user name="fred" password="smith"/>
</basicRegistry>
```

Note: You need to set the **password** attribute of each contained <user> element, even though these passwords will not be used because DataPower will handle the credentials authentication.

For LDAP configuration, see the documentation of your application server.

Note: When moving from development to production or when connecting to an untrusted network, modify the authorized users or the authentication method of the sample pattern, and edit your server configuration to match the pattern. See the authorized-users credentials and authentication method customization note.

2. Configure the authentication-configuration file of your MobileFirst Server: authenticationConfig.xml. (This file is typically found in the server/conf directory of your MobileFirst project.) DataPower will be responsible for authenticating the client, and will use LTPA to communicate the results to MobileFirst Server. However, to represent the user to DataPower you need to define a MobileFirst LTPA form-based authentication realm and LTPA login module. The realm must be of the LTPA authenticator class (com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator) and use the WASLTPAModule login module for LTPA integration of the security framework with WebSphere Application Server. Then, define a security test that uses your realm. You will use this security test in the next step to protect your resources.
The DataPower/server/conf/authenticationConfig.xml file of the sample MobileFirst DataPower integration project includes the required configurations that are explained here.

– WASLTPAModule login module

```
<loginModules>
    ...
    <loginModule name="WASLTPAModule" expirationInSeconds="30">
        <className>com.worklight.core.auth.ext.WebSphereLoginModule</className>
    </loginModule>
</loginModules>
```

– WASLTPARealm realm that uses the WASLTPAModule login module

```
<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>
```

Note: You need to assign values to the **login-page** and **error-page** parameters in the realm definition, even though these parameters will not be used because DataPower will be responsible for sending the login form.

– Security test (DataPowerTest in the following example) that uses the WASLTPARealm realm

```
<securityTests>
    ...
    <customSecurityTest name="DataPowerTest">
        <test realm="wl_antiXSRFRealm" step="1"/>
        <test realm="WASLTPARealm" isInternalUserID="true" step="1"/>
    </customSecurityTest>
<securityTests>
```

For more information about the authentication-configuration file and the related security elements, see the following topics:

– "The authentication configuration file" on page 8-253
– "MobileFirst security configuration" on page 11-89
– "Implementing form-based authenticators" on page 8-274
– "Configuring the MobileFirst LTPA realm" on page 11-92
– "LTPA authenticator" on page 8-290
– "WASLTPAModule login module" on page 8-265
– "Security tests" on page 8-223

- **Protecting your resources** Protect your selected resources (such as adapter procedures or your application) with the form-based LTPA security test that you created in Step 2 on page 6-170 of the server configuration. See "Security configurations of protected resources" on page 8-226.

  For example, the provided sample MobileFirst project contains a JavaScript HTTP adapter with a simple getSecretData procedure (defined in `DataPower/adapter/Protected-impl.js`). This procedure is protected by the DataPowerTest security test that is defined in the sample project (see Step 2 on page 6-170 of the server configuration). The procedure is protected by adding the **securityTest** attribute to the <procedure> element of the getSecretData procedure in the adapter descriptor file (`DataPower/adapter/Protected.xml`):

  ```
  <procedure name="getSecretData" securityTest="DataPowerTest"/>
  ```

- **Configuring DataPower**
  Configure your DataPower appliance with the sample MFP_LTPA_Integration configuration pattern.

  Note: The procedure was tested with DataPower Service Gateway XG45 version 7.0.0.10, and the included DataPower graphical user interface (GUI) references match this product version.

  1. Download the MFP_LTPA_Integration MobileFirst DataPower configuration pattern (`MFP_LTPA_Integration.zip`). The pattern is included in the MobileFirst LTPA DataPower integration package, and can also be downloaded directly here.
  2. Create a new application domain for your DataPower appliance.
  3. Configure the SSL proxy profile that will be used by DataPower to support HTTPS requests from the client application: The sample configuration pattern

uses HTTPS (SSL) to communicate between the mobile device and the DataPower gateway. To support this communication, you need to define an SSL proxy profile. If you do not already have such a profile definition, define an SSL proxy profile in the DataPower WebGUI:

Note: The SSL proxy profile is deprecated beginning with version 7.2.0 of IBM WebSphere DataPower. For information on how to migrate your DataPower configuration from an SSL proxy profile to an SSL client profile, see Migration from SSL Proxy Profile.

a. In the search field of the navigation sidebar, search for `SSL` and select **SSL Proxy Profile** from the results list.

b. In the profile configuration page, select **Add**.

c. Enter the required configuration input for your profile. As part of the configuration you might need to upload an SSL certificate and private keys. The configuration of an SSL profile and the creation of a related certificate are outside the scope of the current documentation. For more information, see the IBM WebSphere DataPower documentation. You can see a sample configuration procedure that uses a self-signed CA root certificate in the following video: Protecting IBM MobileFirst Platform using DataPower.

   Note: Keep in mind that most mobile devices reject self-signed certificates. The workaround is outside the scope of the current documentation.

d. Select **Apply** to apply your changes to the running configuration, and then select **Save Configuration** to save the changes to the startup configuration.

4. Import the MFP_LTPA_Integration pattern:

   a. In the navigation bar of the DataPower WebGUI, select **BluePrint Console**, and make sure that you are logged in to the same application domain that you created in Step 2 on page 6-171.

   b. Select the **Patterns** tab.

   c. Select the import graphic button (next to the **New pattern** button):

   d. In the **Pattern file** field of the Import pattern window, browse to the location of the `MFP_LTPA_Integration.zip` pattern archive file that you downloaded in Step 1 on page 6-171. Then, select **Import** to import the pattern.

5. Deploy the MFP_LTPA_Integration pattern to your DataPower appliance to create a service from the pattern:

   a. Select the **Deploy** button in the pattern heading line.

   b. In the Deploy pattern window, enter the required information for deploying the pattern and creating your service.

      – **Service name** is the service name that you want to display.

      – **Description** is an optional description of your service.

      – **Step 2: Destinations** configures the connection between the DataPower gateway and the MobileFirst Server back-end server:

        - **URL** is the full URL of your MobileFirst Server instance, of the format *`<http or https>`*`://`*`<host_name>`*`:`*`<port>`*`/`*`<project_name>`*.

        - **SSL proxy profile** is the name of the SSL profile that is used by your MobileFirst Server instance to communicate with the DataPower gateway. Configure this field only if your server uses SSL to communicate with the gateway.

      – **Step 3: HTTPS connection** configures the HTTPS (SSL) connection between the DataPower gateway and your client mobile application:

      - **IP address** is the IP address that is the entry point to your application. A value of `0.0.0.0` indicates that any IP address can be used.
      - **Port** is the port number for connecting to your application.
      - **SSL proxy profile** is the SSL proxy profile that you set up in Step 3 on page 6-171. Select this profile from the list.
    - **Step 4: Authenticate with LTPA** and **Step 5** configure the gateway's LTPA keys. Provide the same configuration input in both steps:

      Note: Due to a known issue in versions 7.2 and later of the DataPower firmware, the LTPA steps might not appear when you deploy the pattern. If this is the case, configure the LTPA key manually after your deploy the pattern.
      - **LTPA key file** is your application server's LTPA keys file: `ltpa.keys` (typically found in the `<`*`server_configuration_directory`*`>/servers/worklight/resources/security` directory of your MobileFirst Server).
      - **Key file password** is the password of the MobileFirst Server LTPA keys file. The default keys-file password for WebSphere is `WebAS`.
  c. Select **Deploy pattern** to deploy the pattern to your DataPower appliance and create the service.
  d. Verify that the pattern was deployed successfully: go to the Control Panel of the DataPower WebGUI, select **Multi-Protocol Gateway**, and verify that the gateway's **Op-State** value is `up`.

- **Customization notes**

  Edit the DataPower pattern and MobileFirst configurations, as needed, for your specific needs. Note the following points:

  Changing the authorized-users credentials or the authentication method of the DataPower AAA policy

  By default, the sample MFP_LTPA_Integration pattern checks the user credentials against a `local:///MyAAA.xml` file. The file path is hardcoded in the pattern's `pattern/config/local/MyAAA.xml` file, which also defines test credentials of several authorized users. To use the pattern, your server-configuration file must contain a <basicRegistry> element with one or more of the users that are configured in the pattern, as outlined in Step 1d on page 6-170 of the server configuration.

  You can modify the pattern's authorized-users credentials or authentication method, provided that you adjust the definition of the user-registry element in your server-configuration file to match the pattern. The default configuration is designed for testing during the development stage, and should be modified when connecting to an untrusted network or when moving from development to production. You can modify the configuration, for example, to integrate with an external user register or with any selected security system.

  To modify the authentication method, follow these steps in the DataPower WebGUI:
  1. In the search field of the navigation bar, search for `AAA Policy`. Select the **MFPIntegrationWebHTTPSFormLTPAForm2LTPA** policy from the displayed results.
  2. Select the **Authentication** tab. In the **Method** field, select your preferred authentication method. For example, you can choose **Bind to LDAP server** to bind against a lightweight directory access protocol (LDAP) server. Then, complete the required configuration for your selected method. The pattern can use any valid

authentication method of the DataPower AAA policy, provided that the user identities that are returned by this method also exist in the application server's user registry.

Configuring an HTTP (non-SSL) port

The sample MFP_LTPA_Integration DataPower pattern is configured only for HTTPS (SSL) communication between the mobile device and the DataPower gateway. Therefore, you are required to configure an SSL proxy profile for this interface, as outlined in Step 3 on page 6-171 of the DataPower configuration. However, during development, you can modify the DataPower configuration to open a non-SSL (HTTP) port:

1. In the Control Panel of the DataPower WebGUI, select **Multi-Protocol Gateway**, and look for **Front side settings**.
2. Add an **HTTP Front side handler**, choose a port for HTTP requests, and make sure to check `GET` in the list of allowed methods.

Results

Your MobileFirst application traffic is now protected by a secure DataPower gateway. User authentication is enforced by the DataPower appliance, which forwards the user credentials to MobileFirst Server. When using the sample DataPower MFP_LTPA_Integration pattern, the user credentials are passed as an LTPA token that is contained within a cookie, as demonstrated in the following initial-login, logout, and expired-authentication flow diagrams.

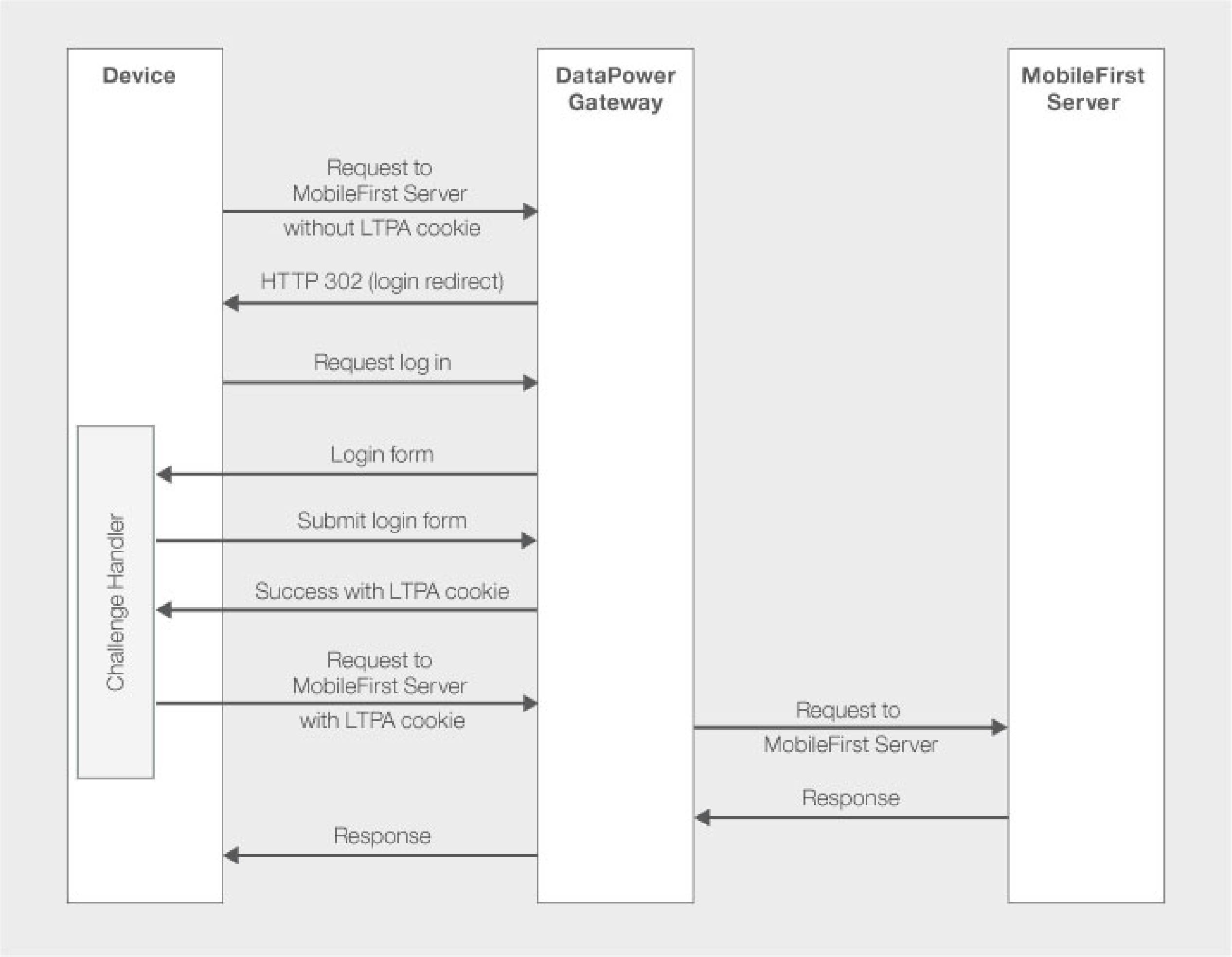

Device
DataPower Gateway
MobileFirst Server
Request to MobileFirst Server without LTPA cookie
HTTP 302 (login redirect)
Request log in
Login form
Submit login form
Success with LTPA cookie
Challenge Handler
Request to MobileFirst Server with LTPA cookie
Request to MobileFirst Server
Response
Response

Figure 6-15. Initial-login flow

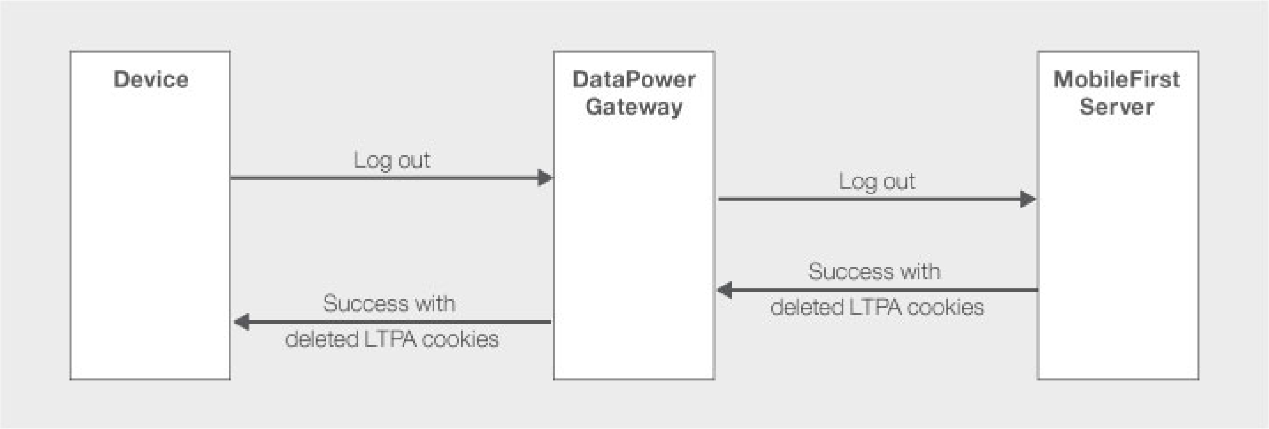

Device
DataPower Gateway
MobileFirst Server
Log out
Log out
Success with deleted LTPA cookies
Success with deleted LTPA cookies

Figure 6-16. Logout flow

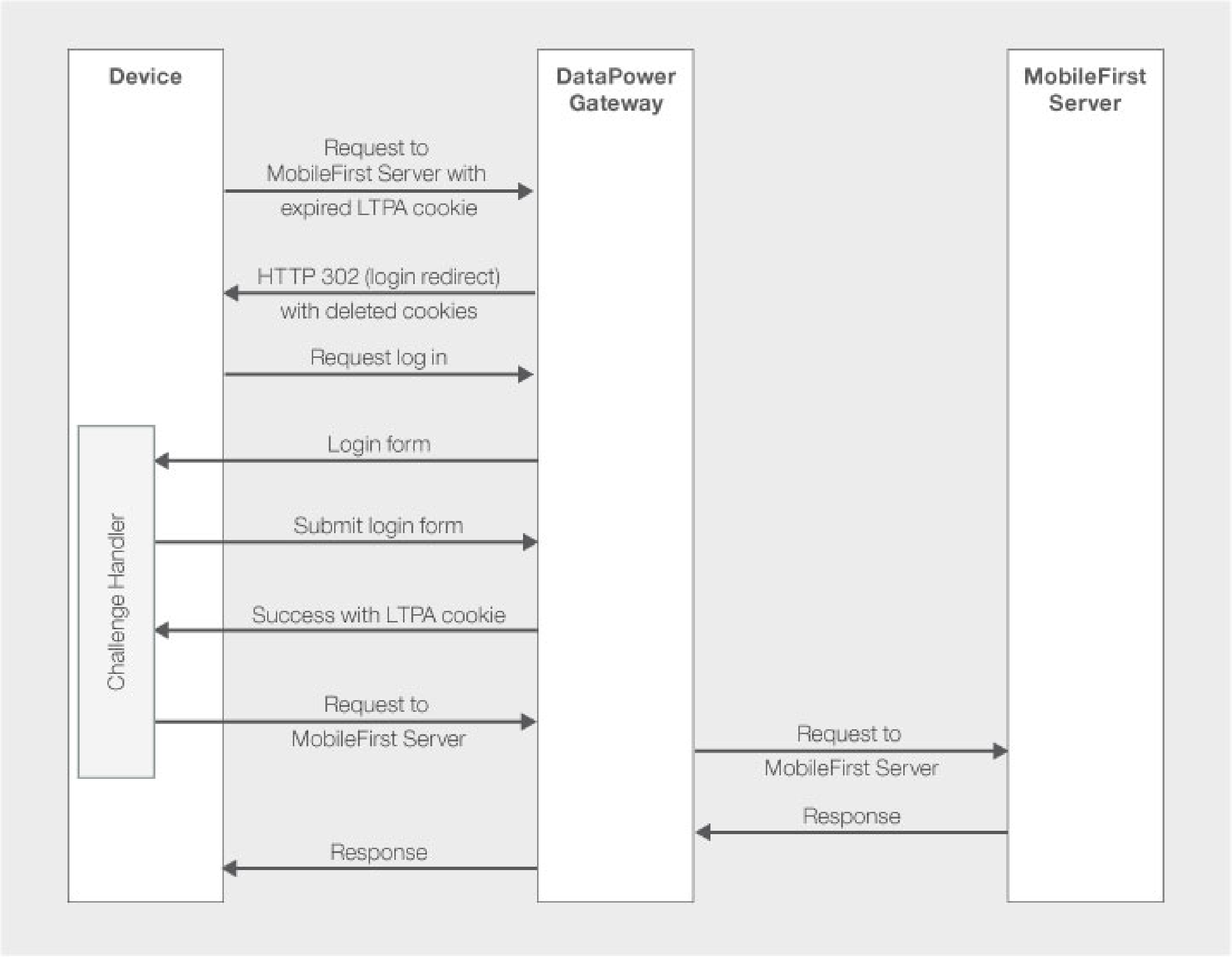

Device
DataPower Gateway
MobileFirst Server
Request to MobileFirst Server with expired LTPA cookie
HTTP 302 (login redirect) with deleted cookies
Request log in
Login form
Challenge Handler
Submit login form
Success with LTPA cookie
Request to MobileFirst Server
Request to MobileFirst Server
Response
Response

Figure 6-17. Expired-authentication flow

Configuring MobileFirst Server to enable TLS V1.2

For MobileFirst 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 MobileFirst Server to enable Transport Layer Security (TLS) V1.2 depend on how MobileFirst Server connects to devices.

If MobileFirst 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 MobileFirst Server communicates directly with devices, the steps to configure MobileFirst 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 have one of the following JRE versions, depending on your version of IBM MobileFirst Platform Foundation for iOS.

   For MobileFirst Server V7.1 or later:
   Use one of the following JRE versions.
   - Oracle JRE 1.7.0_75 or later
   - Oracle JRE 1.8.0_31 or later

   For MobileFirst Server V7.0 or earlier:
   Use Oracle JRE 1.7.0_75 or later.
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 have one of the following versions, depending on your version of IBM MobileFirst Platform Foundation for iOS.

     For MobileFirst Server V7.1 or later:
     Use one of the following JRE versions.
     - Oracle JRE 1.7.0_75 or later
     - Oracle JRE 1.8.0_31 or later

     For MobileFirst Server V7.0 or earlier:
     Use Oracle JRE 1.7.0_75 or later.
2. If you use an IBM Java SDK, edit the `server.xml` file.
   1. Add the following line:

      ```
      <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="SSL_TLSv2"/>
      ```
   2. 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 MobileFirst Server to enable ATS for iOS 9

App Transport Security (ATS) is a feature introduced to improve secure communications between the App and the Web Services. This feature is available in iOS version 9 or above.

About this task

The steps to configure MobileFirst Server to enable App Transport Security is explained in How to enable App Transport Security for iOS 9 in IBM MobileFirst Platform Server.

For more information on **App Transport Security (ATS)**, see this technote.

Configuring SSL between MobileFirst adapters and back-end servers by using self-signed certificates

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

Procedure

1. Check the configuration in the `worklight.properties` file. The configuration might look like this example (the password will be different for each project):

```
####################################################################################################################
#    MobileFirst SSL keystore
####################################################################################################################
#SSL certificate keystore location.
ssl.keystore.path=conf/mfp-default.keystore
#SSL certificate keystore type (jks or PKCS12)
ssl.keystore.type=jks
#SSL certificate keystore password.
ssl.keystore.password=oW523Mes0b24lqAXc5F7
```

2. Make sure that the keystore file exists in the `server/conf` folder of the MobileFirst 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 MobileFirst keystore.
5. Restart the MobileFirst 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 JKS
```

   Note: The **First and Last Name** field contains your server URL, which you use in theadapter.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 certificate from the key store
    <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 MobileFirst Server mfp-default.keystore file in the server/conf directory of the MobileFirst project:

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

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

```
keytool -list -keystore mfp-default.keystore
```

Configuring SSL by using untrusted certificates

Making SSL work between instances of IBM MobileFirst Platform Server 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.

Support for X.509 certificates comes from the individual platforms, not from IBM MobileFirst Platform Foundation for iOS. 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 MobileFirst 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.

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 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 -r
```

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 MobileFirst 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 6-185.

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, see "Updating your keystore and Liberty profile configuration to use a certificate chain" on page 6-187.

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 6-50. 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: `basicConstraints = CA:TRUE` For more information, see "Self-signed certificates versus self-signed CAs" on page 6-181. Ensure that the certificate is in `PEM` format. Ensure that the certificate has a `.crt` file extension.
`"errorCode":"UNRESPONSIVE_HOST","errorMsg":"The service is currently not available."`	This error usually indicates an SSL handshake failure. The client cannot establish trust for the server certificate. 1. Ensure that you installed the server's root CA on the client device. For more information, see "Establishing trust on the client" on page 6-181. 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 6-181. 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 6-182. 2. Ensure that the CRL URL is valid and reachable. For more information, see "CRL support" on page 6-182. 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 6-182.

Table 6-50. 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" on page 6-185.
`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 6-181. • Ensure that the server sends the complete certificate chain and in the right order. For more information, see "Handling the certificate chain" on page 6-181.

Related tasks:

"Configuring SSL for Liberty profile" on page 6-277
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

Specifying a trusted SSL certificate

While the certificates from several certificate authorities (CA) are already recognized and trusted by the MobileFirst Platform Command Line Interface for iOS, you can authorize an untrusted CA by adding its certificate path as an environment variable.

About this task

The Node.js system contains several CAs whose signed certificates are trusted by default. If you try to connect a server with a certificate from an untrusted source by using the MobileFirst Platform Command Line Interface for iOS, the operating system displays the following message:

```
Failed to retrieve runtime information: CERT_UNTRUSTED
```

This can happen when you are using a self-signed certificate, one from a private certificate authority, or a public CA that is not recognized as a trusted source by Node.js.

You can add up to 10 certificates to be trusted to your environment variables. To add a certificate, complete the following steps:

Procedure

1. Copy the path to the certificate file that you want to add to the list of trusted certificates. The certificate file must be in the `.PEM` format.
2. Add the environment variable.
   - For a Unix-based operating system, complete the following steps:
     a. Open a terminal window.
     b. Enter the following command to add the environment variable:
     ```
     $ set MFP_HTTPS_SSL_TRUSTSTORE_0=path_to_certificate/certificate_file.cer
     ```
     Where *path_to_certificate* is the path to your certificate, and *certificate_file* is the filename of the certificate.
     c. Enter the following command to integrate the environment variable:
     ```
     $ export MFP_HTTPS_SSL_TRUSTSTORE_0
     ```

     Note: These steps must be completed for the untrusted certificates whenever you start a new shell.
   - For the Windows operating system, complete the following steps:
     a. Open your environment variables by selecting **Start** > **Control Panel** > **System and Security** > **Advanced Settings** > **Environment Variables...**.

     Important: Use caution when you add the new Environment Variables. Accidentally modifying an existing Environment Variable can cause problems for other programs on your system.
     b. In System Variables, select **New...**
     c. Enter `MFP_HTTPS_SSL_TRUSTSTORE_0` for your Variable name.
     d. Enter `/path_to_certificate.cer` for your Variable value.
     e. Restart the operating system.
3. Repeat these steps for any other untrusted certificates that you need to add. You can add up to nine more certificates by changing the 0 in the variable to a digit that is 1-9, inclusive.

Installing the root CA on iOS

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

About this task

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

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

Procedure

1. Ensure that the root CA is in `PEM` file format and has a `.crt` file extension. Convert as needed.
2. Run the following command to view the certificate details.
```
openssl x509 -in certificate.crt -text -noout
```

3. Ensure that the certificate is of version X.509 v3. The certificate details must show `Version 3`.

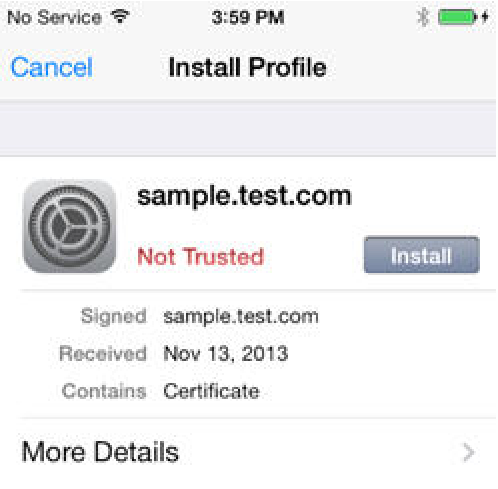
   Note: The following `openssl` flag generates X.509 v3 certificates:

   ```
   -reqexts v3_req
   ```
4. Ensure that the certificate is a certificate authority. The certificate details must show `X509v3 Basic Constraints: CA:TRUE`

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

   ```
   -extensions v3_ca
   ```
5. To download the certificate file on the device, send it as an email attachment or host it on a secure website.

   Note: Do not install the server certificate by accessing the protected resource directly from your browser. This action imports the certificate only into the browser space and not into the device system truststore.
6. After you have the certificate file on the device, click the file to allow the iOS system to install the certificate.



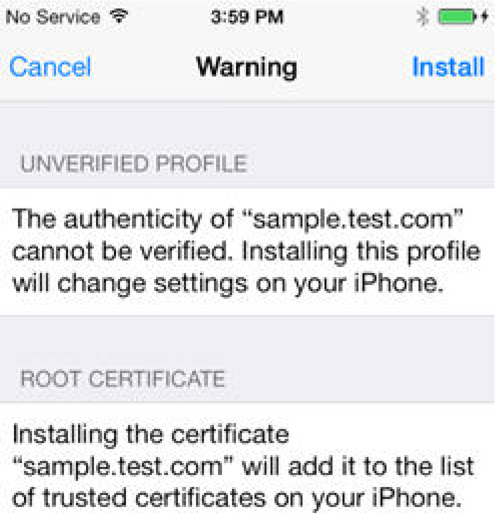
sample.test.com

Not Trusted

Signed sample.test.com

Received Nov 13, 2013

Contains Certificate



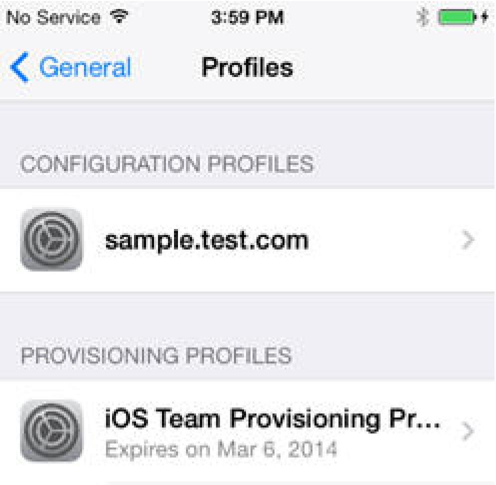
UNVERIFIED PROFILE

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

ROOT CERTIFICATE

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

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



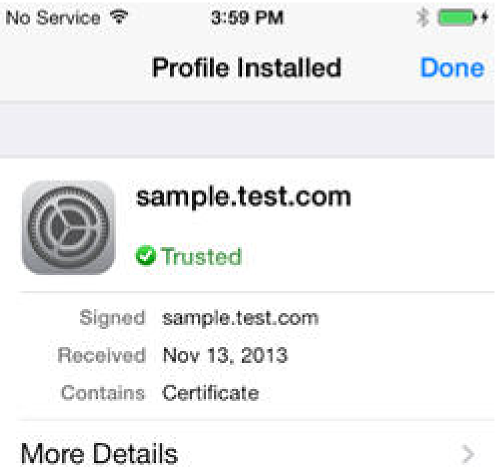
CONFIGURATION PROFILES

sample.test.com

PROVISIONING PROFILES

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

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



sample.test.com
✓ Trusted

Signed sample.test.com
Received Nov 13, 2013
Contains Certificate

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.

Handling MySQL stale connections

You can configure your application server to avoid MySQL timeout issues.

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

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

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

You can configure your application server so as to avoid this exception if you use MySQL databases.

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 a 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. 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.

Edit the `server.xml` file and for every `<dataSource>` element (runtime 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. 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.

1. Log in to the WebSphere Application Server console.
2. Select **Resources** > **JDBC** > **Data sources**.
3. For each MySQL data source:
   a. Click the data source.
   b. Select **Connection pool properties** under **Additional Properties**.
   c. Modify the value of the **Aged timeout** property. The value must be lower than the MySQL `wait_timeout` system variable so that the connections are purged before MySQL closes these connections.
   d. Click **OK**.

Handling DB2 and Oracle stale connections

You can configure your application server to avoid DB2 and Oracle timeout issues.

A StaleConnectionException is an exception that is generated by the WebSphere Application Server or the WebSphere Application Server Liberty profile database connection code when a JDBC driver returns an unrecoverable error from a connection request or operation. The StaleConnectionException is raised when the database vendor issues an exception to indicate that a connection currently in the connection pool is no longer valid. This exception can happen for many reasons. The most common cause of StaleConnectionException is due to retrieving connections from the database connection pool and finding out that the connection has timed out or dropped when it was unused for a long time.

You can configure your application server to avoid this exception.

WebSphere Application Server full profile configuration

To minimize the stale connection issues, check the connection pools configuration on each data source in WebSphere Application Server administration console.

1. Log in to the WebSphere Application Server administration console.
2. Select **Resources** > **JDBC Providers** > ***database_jdbc_provider*** > **Data Sources** > ***your_data_source*** > **Connection pool properties**.
3. Set the **`Minimum connections`** value to `0`.
4. Set the **`Reap time`** value to be lesser than the **`Unused timeout`** value.
5. Make sure that the **`Purge policy`** property is set to `EntirePool` (default).

For more information, see Connection pool settings.

WebSphere Application Server Liberty profile configuration

Edit the `server.xml` file. For every `<dataSource>` element, add a `<connectionManager>` element with the **`agedTimeout`** attribute:

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

For example:

```
<dataSource jndiName="jndi_name" transactional="false">
  <connectionManager agedTimeout="10m"/>
  ...
</dataSource>
```

The timeout value depends mainly on the number of opened connections in parallel but also on the minimum and maximum number of the connections in the pool. Hence, you must tune the different `connectionManager` attributes to identify the most adequate values. For more information about the `connectionManager` element, see Liberty: Configuration elements in the server.xml file .

Managing the DB2 transaction log size

When you deploy an application that is at least 40 MB with IBM MobileFirst Platform Operations 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 MobileFirst 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 MobileFirst 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 MobileFirst Platform Operational Analytics

The IBM MobileFirst Platform Operational Analytics is delivered as two separate WAR files. For convenience in deploying on WebSphere Application Server or WebSphere Application Server Liberty, the IBM MobileFirst Platform Operational Analytics is also delivered as an EAR file that contains the two WAR files.

Note: Do not install more than one instance of IBM MobileFirst Platform Operational Analytics on a single host machine. For more information about managing your cluster, see the Elasticsearch documentation.

When you develop within MobileFirst Platform Command Line Interface for iOS, the WAR files that contain the IBM MobileFirst Platform Operational Analytics are automatically deployed. The MobileFirst Server forwards data to the MobileFirst tools with no additional required configurations.

The analytics WAR and EAR files are included with the MobileFirst Server installation. For more information, see “Distribution structure of MobileFirst Server” on page 6-42.

The following sections describe the required steps for successfully deploying the WAR file to the application server.

When you deploy the WAR file, the MobileFirst Analytics Console is available at:

```
http://<hostname>:<port>/analytics/console
```

Example:

```
http://localhost:9080/analytics/console
```

Installing MobileFirst Operational Analytics with Ant tasks

Learn how to use Ant tasks to deploy MobileFirst Operational Analytics to your application server.

Before you begin

You must run the Ant task on the computer where the application server is installed, or the Network Deployment Manager for WebSphere Application Server Network Deployment. If you want to start the Ant task from a computer on which

MobileFirst Server is not installed, you must copy the file *mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar to that computer.

Note: The *mf_server_install_dir* placeholder is the directory where you installed MobileFirst Server.

For more information, see "Production deployment and clustering" on page 13-71.

Procedure

1. Edit the Ant script that you use later to deploy MobileFirst Operational Analytics WAR files.
   a. Review the sample configuration files in "Sample configuration files for MobileFirst Operational Analytics" on page 15-71.
   b. Replace the placeholder values with the properties at the beginning of the file.

      Note: The following special characters must be escaped when they are used in the values of the 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 section of the Apache Ant Manual.
      - The ampersand character (&) must be written as &amp;, unless you explicitly want to reference an XML entity.
      - Double quotation marks (") must be written as &quot;, except when it is inside a string that is enclosed in single quotation marks.
2. If you install a cluster of nodes on several servers:
   a. You must uncomment the property wl.analytics.masters.list, and set its value to the list of host name and transport port of the master nodes. For example:

      ```
      node1.mycompany.com:96000,node2.mycompany.com:96000
      ```
   b. Add the attribute **mastersList** to the **elasticsearch** elements in the tasks installanalytics, updateanalytics, and uninstallanalytics.

      Note: If you install on a cluster on WebSphere Application Server Network Deployment, and you do not set the property, the Ant task computes the data end points for all the members of the cluster at the time of installation, and set the masternodes JNDI property to that value.

      Alternatively, you can configure the cluster after installation. For more information, see "Properties and configurations" on page 13-90.
3. To deploy the WAR files, run the following command:

   ```
   ant -f configure-appServer-analytics.xml install
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts. This installs a node of MobileFirst Operational Analytics, with the default type master and data, on the server, or on each member of a cluster if you install on WebSphere Application Server Network Deployment.
4. Save the Ant file. You might need it later to apply a fix pack or perform an upgrade. For more information, see "Applying a fix pack to IBM MobileFirst Platform Operational Analytics" on page 7-85.

   If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.

Note: If you add a node to a cluster of MobileFirst Operational Analytics, you must update the `analytics/masternodes` JNDI property, so that it contains the ports of all the master nodes of the cluster.

What to do next

- "Securing the Operational Analytics server" on page 13-69
- To configure Analytics, see "Properties and configurations" on page 13-90

See also :

- "Ant tasks for installation of MobileFirst Operational Analytics" on page 15-54

Installing IBM MobileFirst Platform Operational Analytics manually

You can manually install IBM MobileFirst Platform Operational Analytics, by running an Analytics EAR file on your application server.

Installing IBM MobileFirst Platform Operational Analytics for WebSphere Application Server Liberty

You can install the IBM MobileFirst Platform Operational Analytics for WebSphere Application Server Liberty.

About this task

The following steps describe how to install and run the Analytics EAR file on WebSphere Application Server Liberty.

The IBM MobileFirst Platform Operational Analytics is protected with role-based security, so you must bind the security roles to the application to be able to access the console.

Procedure

1. Add the Analytics EAR file to the `apps` folder of your WebSphere Application Server Liberty application server.
2. Modify the `server.xml` file to set the class loading delegation and bind the security roles.

```
<basicRegistry id="worklight" realm="worklightRealm">
  <user name="demo" password="demo"/>
  <user name="monitor" password="demo"/>
  <user name="deployer" password="demo"/>
  <user name="operator" password="demo"/>
  <user name="admin" password="admin"/>
</basicRegistry>

<application location="analytics.ear"
    name="analytics-ear"
    type="ear">
  <application-bnd>
    <security-role name="worklightadmin">
       <user name="admin"/>
    </security-role>
    <security-role name="worklightdeployer">
       <user name="deployer"/>
    </security-role>
    <security-role name="worklightmonitor">
       <user name="monitor"/>
    </security-role>
    <security-role name="worklightoperator">
```

```
      <user name="operator"/>
    </security-role>
  </application-bnd>
</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 and view the console in the browser.

   ```
   http://localhost:9080/analytics/console
   ```

Results

The analytics console is deployed and can now be viewed in the browser.

Installing IBM MobileFirst Platform Operational Analytics for WebSphere Application Server

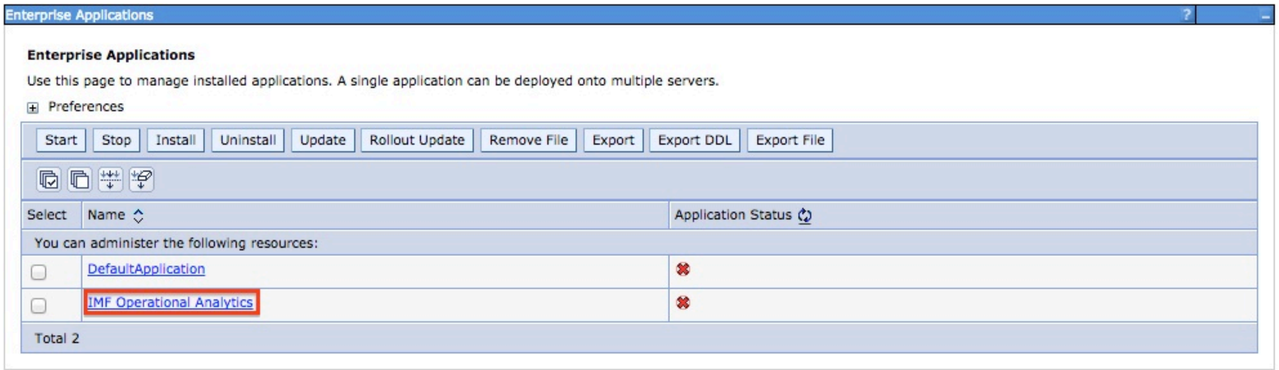
You can install the IBM MobileFirst Platform Operational Analytics for WebSphere Application Server.

About this task

The following steps describe how to install and run the Analytics EAR file on WebSphere Application Server. If you are installing the individual WAR files on WebSphere Application Server, follow only steps 2 - 6 on the `analytics-service` WAR file after you deploy both WAR files. The class loading order must not be altered on the `analytics-ui` WAR file.

Procedure

1. Deploy the EAR file to the application server, but do not start it. For more information about the analytics files, see "Distribution structure of MobileFirst Server" on page 6-42. For more information about how to install an EAR file on WebSphere Application Server, see Installing enterprise application files with the console.
2. Select the **IMF Operational Analytics** application from the **Enterprise Applications** list.



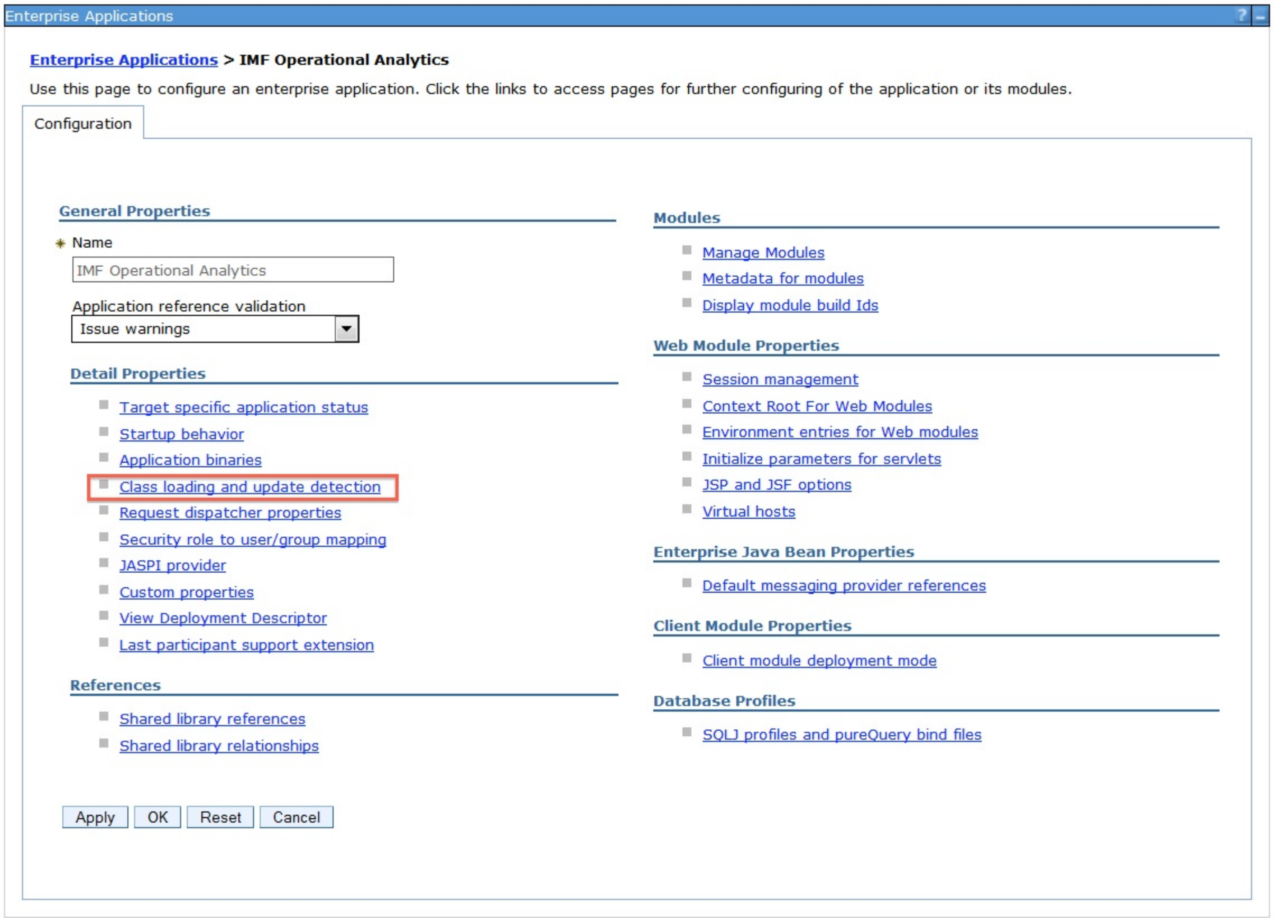
Enterprise Applications

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

⊞ Preferences

Select	Name	Application Status
You can administer the following resources:		
□	DefaultApplication	✖
□	IMF Operational Analytics	✖
Total 2		

3. Click **Class loading and update detection**.



Enterprise Applications

Enterprise Applications > IMF Operational Analytics

Use this page to configure an enterprise application. Click the links to access pages for further configuring of the application or its modules.

Configuration

General Properties

* Name

IMF Operational Analytics

Application reference validation

Issue warnings

Detail Properties

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

References

- Shared library references
- Shared library relationships

Modules

- Manage Modules
- Metadata for modules
- Display module build Ids

Web Module Properties

- Session management
- Context Root For Web Modules
- Environment entries for Web modules
- Initialize parameters for servlets
- JSP and JSF options
- Virtual hosts

Enterprise Java Bean Properties

- Default messaging provider references

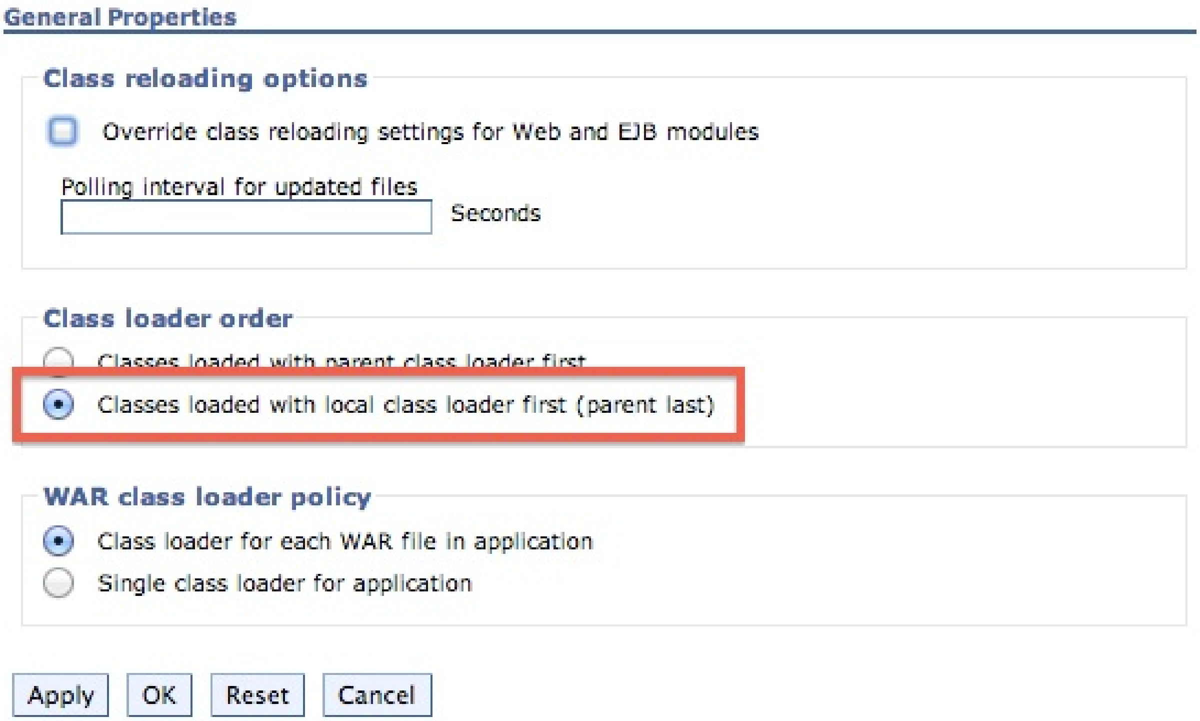
Client Module Properties

- Client module deployment mode

Database Profiles

- SQLJ profiles and pureQuery bind files

4. Set the class loading 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

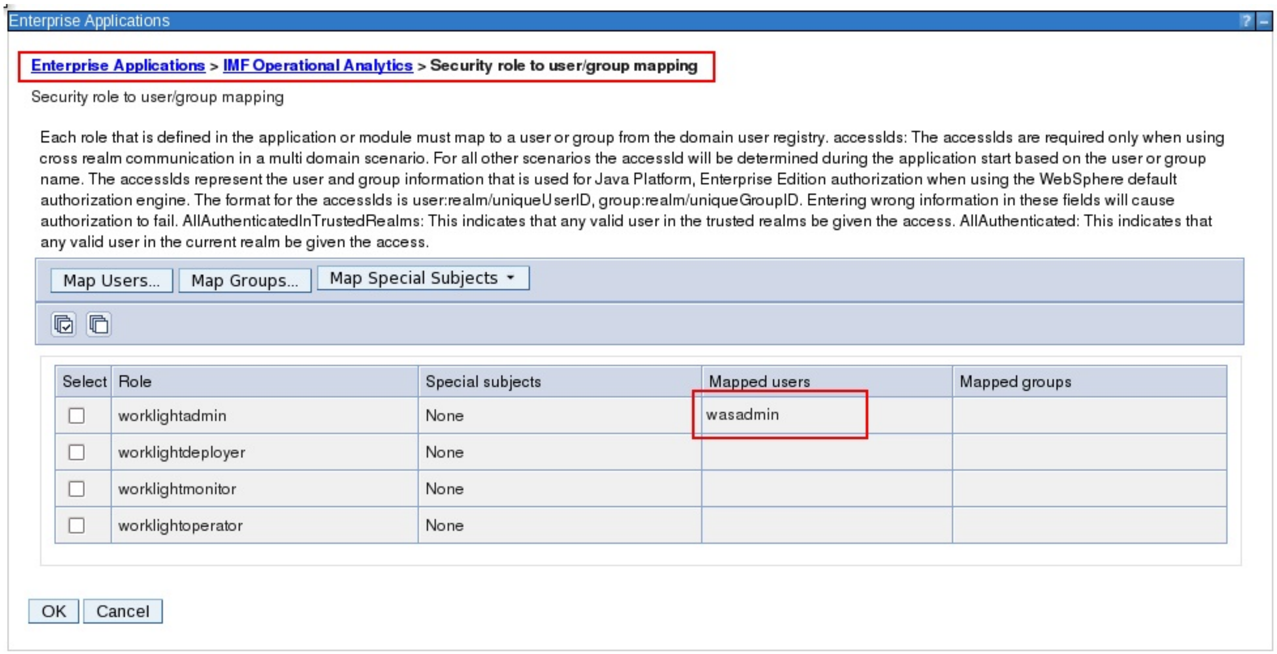
● Classes loaded with local class loader first (parent last)

WAR class loader policy

● Class loader for each WAR file in application

○ Single class loader for application

5. Click **Security role to user/group mapping** to map the admin user.



Enterprise Applications

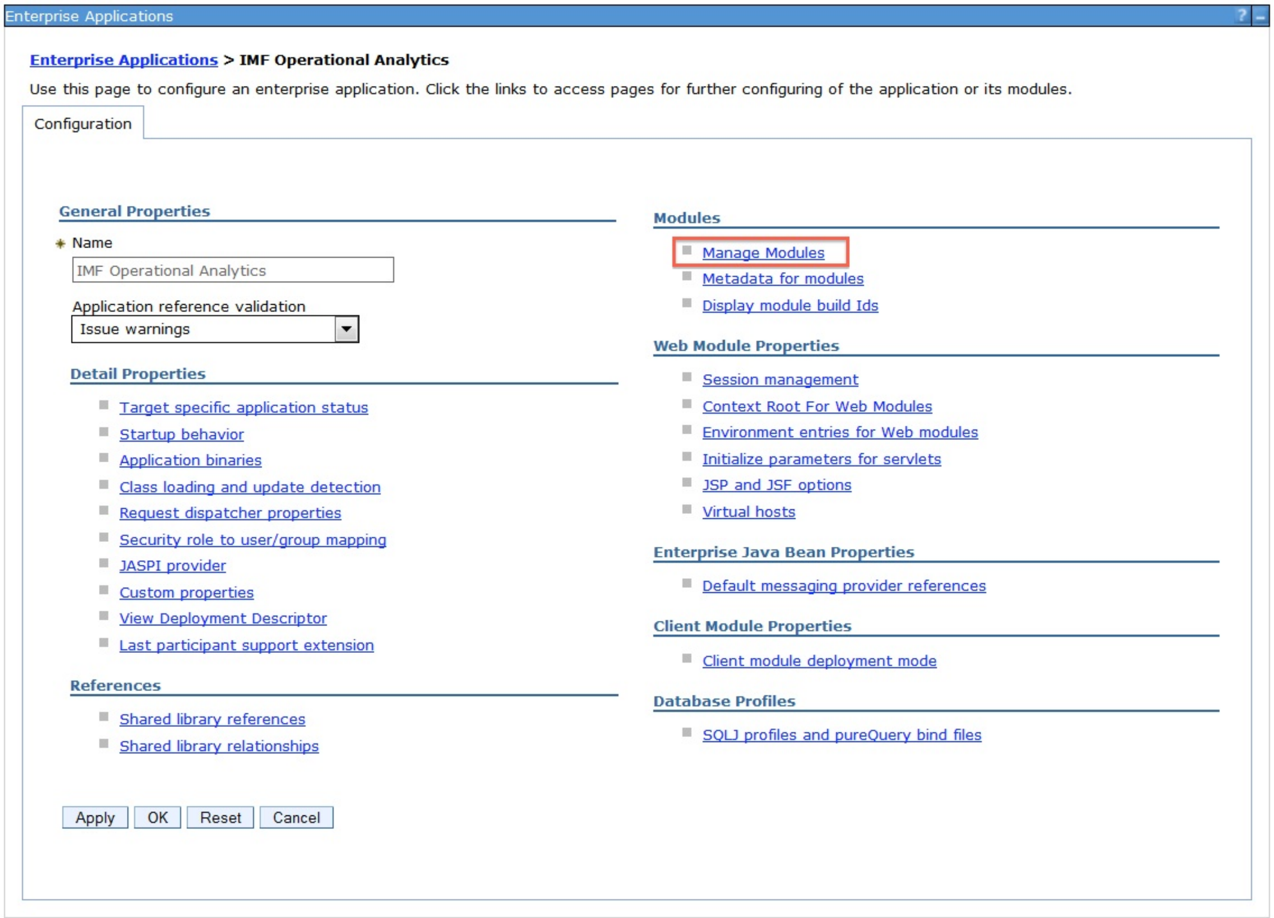
Enterprise Applications > IMF Operational Analytics > Security role to user/group mapping

Security role to user/group mapping

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

Select	Role	Special subjects	Mapped users	Mapped groups
☐	worklightadmin	None	wasadmin	
☐	worklightdeployer	None		
☐	worklightmonitor	None		
☐	worklightoperator	None		

6. Click **Manage Modules**.



Enterprise Applications > **IMF Operational Analytics**

Use this page to configure an enterprise application. Click the links to access pages for further configuring of the application or its modules.

General Properties

* Name
IMF Operational Analytics

Application reference validation
Issue warnings

Detail Properties

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

References

- Shared library references
- Shared library relationships

Modules

- Manage Modules
- Metadata for modules
- Display module build Ids

Web Module Properties

- Session management
- Context Root For Web Modules
- Environment entries for Web modules
- Initialize parameters for servlets
- JSP and JSF options
- Virtual hosts

Enterprise Java Bean Properties

- Default messaging provider references

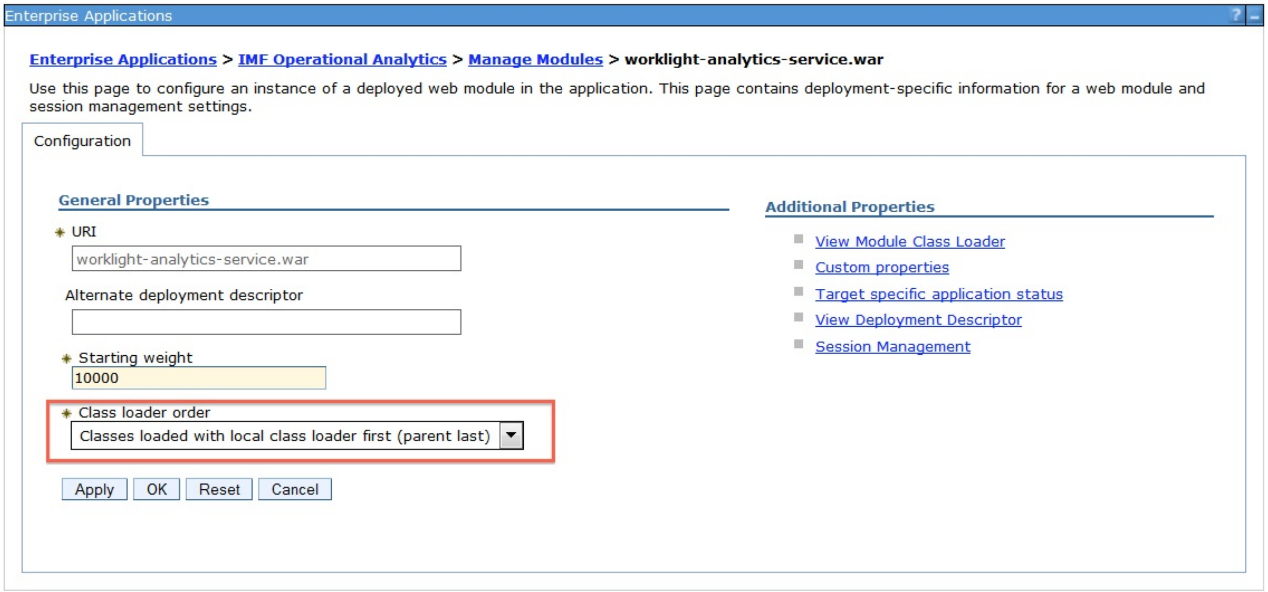
Client Module Properties

- Client module deployment mode

Database Profiles

- SQLJ profiles and pureQuery bind files

7. Select the **worklight-analytics-service** module and change the class loading order to **parent last**.



Enterprise Applications > **IMF Operational Analytics** > **Manage Modules** > **worklight-analytics-service.war**

Use this page to configure an instance of a deployed web module in the application. This page contains deployment-specific information for a web module and session management settings.

General Properties

* URI
worklight-analytics-service.war

Alternate deployment descriptor

* Starting weight
10000

* Class loader order
Classes loaded with local class loader first (parent last)

Additional Properties

- View Module Class Loader
- Custom properties
- Target specific application status
- View Deployment Descriptor
- Session Management

8. Start the **IMF Operational Analytics** application and go to the link in the browser.

```
http://<hostname>:<port>/analytics-service/data
```

Results

The analytics EAR file is now ready to accept incoming analytics data.

Installing IBM MobileFirst Platform Operational Analytics for Apache Tomcat

To configure Apache Tomcat for MobileFirst Operational Analytics manually, you must copy WAR files to Tomcat, edit the `server.xml` file, add the required users, and start Tomcat..

1. Edit the file *tomcat_install_dir*/conf/tomcat-users.xml.
   1. Add the security roles:

      ```
      <role rolename="worklightadmin"/>
      <role rolename="worklightdeployer"/>
      <role rolename="worklightmonitor"/>
      <role rolename="worklightoperator"/>
      ```

   2. Add roles to the selected users, for example:

      ```
      <user name="admin" password="admin" roles="worklightadmin"/>
      ```

      Note: You can define the set of users as described in the Apache Tomcat documentation, at Realm Configuration HOW-TO.
2. Edit the file *tomcat_install_dir*/conf/server.xml:
   1. Uncomment the following element, which is initially commented out, or add the element if it is not present:

      ```
      <!-- SingleSignOn valve, share authentication between web applications. Documentation at: htt
      <Valve className ="org.apache.catalina.authenticator.SingleSignOn"/>
      ```

   2. Declare the MobileFirst Operational Analytics Services and the MobileFirst Operational Analytics Console, as well as a user registry:

      ```
      <!-- Declare the MobileFirst Operational Analytics Service application. -->
      <Context docBase ="analytics-service" path ="/analytics-service">

        <!-- Declare the JNDI environment entries for the MobileFirst Operational Analytics Service
        <Environment name ="analytics/masternodes" value ="hostname:9600" type ="java.lang.String" 
      </Context>

      <!-- Declare the MobileFirst Operational Analytics Console application. -->
      <Context docBase ="analytics" path ="/analytics"></Context>

      <!-- Declare the user registry for the IBM Application Center. The MemoryRealm recognizes the
      <Realm className ="org.apache.catalina.realm.MemoryRealm"/>
      ```

      Note: You must change the value of the JNDI property `analytics/masternodes` to reflect the host name or IP address of the machine on which you are installing MobileFirst Operational Analytics. By default, the port for ElasticSearch is 9600.
3. Copy the `analytics-service.war` file to *tomcat_install_dir*/webapp by using the following commands, according to your operating system.
   - On UNIX and Linux systems:

     ```
     cp product_install_dir/Analytics/analytics-service.war tomcat_install_dir/webapps
     ```

   - On Windows systems:

     ```
     copy /B product_install_dir\Analytics\analytics-service.war tomcat_install_dir\webapps\analyti
     ```
4. Copy the `analytics-ui.war` file to *tomcat_install_dir*/webapp and rename the file to `analytics.war` by using the following commands, according to your operating system.

- On UNIX and Linux systems:

  ```
  cp product_install_dir/Analytics/analytics-service.war tomcat_install_dir/webapps/analytics.
  ```

- On Windows systems:

  ```
  copy /B product_install_dir\Analytics\analytics-ui.war tomcat_install_dir\webapps\analytics.
  ```

Configuring the MobileFirst Server for the IBM MobileFirst Platform Operational Analytics

You must configure the MobileFirst Server for the IBM MobileFirst Platform Operational Analytics.

About this task

The following steps describe how to configure the MobileFirst Server for the IBM MobileFirst Platform Operational Analytics.

Procedure

1. In the `worklight.properties` file, set the `wl.analytics.url` property to point to the deployed WAR file.

   ```
   wl.analytics.url=http://<hostname>:<port>/analytics-service/data
   ```

   For example, if the Liberty server is at `host.ibm.com` on port 8080, then the `wl.analytics.url` property is as follows:

   ```
   wl.analytics.url=http://host.ibm.com:8080/analytics-service/data
   ```

2. In the `worklight.properties` file, set the `wl.analytics.username` and the `wl.analytics.password` properties.
3. Optional: If you want to access the MobileFirst Analytics Console from the MobileFirst Operations Console, set the `wl.analytics.console.url` property in the `worklight.properties` file.

   ```
   wl.analytics.console.url=http://<hostname>:<port>/analytics/console
   ```

   For example, if the Liberty server is at `host.ibm.com` on port 8080 and the context root is `analytics`, then the `wl.analytics.console.url` property is as follows:

   ```
   wl.analytics.console.url=http://host.ibm.com:8080/analytics/console
   ```

4. Optional: If you want to send logging information from a package other than the `worklight` package, in the `worklight.properties` file, set the `wl.analytics.logs.packages` property to the relevant package name, for example:

   ```
   #Capture logs from the user defined packages (by default logs captured
   only from the "com.worklight" package)
   ```

   ```
   wl.analytics.logs.package=my.package.com
   ```

Results

The MobileFirst Server now forwards data to the IBM MobileFirst Platform Operational Analytics.

Note: All properties in the `worklight.properties` file can also be set by using JNDI. For more information about JNDI settings, see “Configuration of MobileFirst applications on the server” on page 11-50.

Installing the MobileFirst Data Proxy

You install the MobileFirst Data Proxy to serve as a proxy between your MobileFirst Server and your Cloudant database.

Planning the installation of MobileFirst Data Proxy

Before you install the MobileFirst Data Proxy, you must plan your installations and verify the prerequisites for your system.

Installation overview of the MobileFirst Data Proxy

With the MobileFirst Data Proxy, the data from your mobile applications can be saved on the server side.

To develop applications with this feature, use the MobileFirst Data Proxy SDK that is documented at "Storing mobile data in Cloudant" on page 8-151.

The MobileFirst Data Proxy service requires an installed IBM MobileFirst Platform runtime, and access to a Cloudant database.

IBM MobileFirst Platform Foundation for iOS bundles a limited-use Virtual Server entitlement to IBM MobileFirst Platform Cloudant Data Layer Local Edition. This Virtual Server, or node, provides development and test with full API support, and tools. This node can be used for production with appropriate planning for availability, performance, and backup. For more information about the use limitation, see the IBM MobileFirst Platform Foundation for iOS license. By purchasing additional entitlements to IBM MobileFirst Platform Cloudant Data Layer Local Edition, customers can cluster multiple nodes together and gain horizontal and geographic scalability, fault tolerance, and continuous availability. You might want to deploy in a clustered topology for applications that require the availability, elasticity, and reach of possibly massive amounts of mobile data and devices. It is best suited for applications that require an operational data store to handle a massively concurrent mix of low-latency reads and writes.

Attention: You can install only one instance of an MobileFirst Data Proxy in an application server. That instance can authenticate incoming requests with only one MobileFirst project runtime.

Installation prerequisites for the MobileFirst Data Proxy

Review the system requirements, and perform the required installations of MobileFirst Server and the Cloudant database before installing the MobileFirst Data Proxy. Some restrictions apply about the type of application server that you can use.

For more information about the supported hardware and pre-required software, see "System requirements" on page 2-9.

Before installing the MobileFirst Data Proxy, you must perform the following actions:

- Install an instance of MobileFirst Server. For more information about the installation process, see "Installing MobileFirst Server" on page 6-3.
- Deploy a project WAR file. For more information about the deployment process, see "Deploying the project WAR file" on page 11-5.

- Install IBM MobileFirst Platform Cloudant Data Layer Local Edition, or get access to the Cloudant database. For more information about the installation process, see the IBM MobileFirst Platform Cloudant Data Layer Local Edition user documentation.

Application server restrictions

You can install the MobileFirst Data Proxy on the following application servers:

- WebSphere Application Server Liberty profile V8.5.5.0 and later
- WebSphere Application Server full profile
- WebSphere Application Server Network Deployment

You cannot install MobileFirst Data Proxy on the following application server:

- Apache Tomcat
- WebSphere Application Server Liberty profile V8.5.0.x

File System prerequisites for MobileFirst Data Proxy

Review the file system privileges and the specific rights for WebSphere Application Server and WebSphere Application Server Network Deployment that you must have before you install the MobileFirst Data Proxy.

The file system prerequisites are the same as for installing the MobileFirst Server, which are described in "File system prerequisites" on page 6-5.

In addition, to install on WebSphere Application Server or WebSphere Application Server Network Deployment, you must have the right to create files in the WebSphere Application Server installation directory: `<`*`was_install_dir`*`>/lib/ext`.

For WebSphere Application Server Network Deployment, you must do this operation on every node of the WebSphere Application Server Network Deployment cell, even if they do not run the MobileFirst Data Proxy service. This is required to install a Trust Association Interceptor (TAI) that is declared at the cell level.

Installing and configuring the MobileFirst Data Proxy

You can choose to install the MobileFirst Data Proxy with Ant tasks, or manually. For more information about the procedures to follow for each case, see the appropriate topics in this section.

Installing the MobileFirst Data Proxy with Ant tasks

Learn about the Ant tasks that you can use to install the MobileFirst Data Proxy.

Before you begin

Make sure that Cloudant is installed and running. It can be on the same computer, or a different computer. The Ant task verifies the connectivity to Cloudant before proceeding with the installation.

To deactivate that verification, see "Ant tasks for installation of MobileFirst Data Proxy" on page 15-49.

You must install the MobileFirst Server, as described in "Installing MobileFirst Server" on page 6-3, and deploy a project WAR file, as described in "Deploying the project WAR file" on page 11-5.

You must have the URL of the deployed project WAR file to complete the installation of the MobileFirst Data Proxy. If you want to start the Ant task from a computer on which MobileFirst Server is not installed, you must copy the following files on that computer:

- *mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar
- *mf_server_install_dir*/Datastore/imf-data-proxy.jar
- *mf_server_install_dir*/WorklightServer/external-server-libraries/*

About this task

Procedure

1. On WebSphere Application Server Network Deployment, you must install the Trust Association Interceptor (TAI) manually on every node of the WebSphere Application Server Network Deployment cell. For more information on the installation instructions, see "Installing the MobileFirst OAuth Trust Association Interceptor (TAI)" on page 6-205.
2. Review the sample configuration files, and copy the Ant file that corresponds to your application server.

   The following list of sample configuration files are in *product_install_dir*/ Datastore/configuration-samples:
   - configure-liberty.xml: to install on a Liberty server.
   - configure-was.xml: to install on a WebSphere Application Server stand-alone server.
     - configure-wasnd-cluster.xml: to install on WebSphere Application Server Network Deployment, on a cluster.
     - configure-wasnd-server.xml: to install on WebSphere Application Server Network Deployment, on a managed server.
3. Edit the Ant file and replace the placeholder values for the properties at the beginning of the file.
4. Run the following command to install the MobileFirst Data Proxy:

   ```
   ant -f configure-<appserver>.xml install
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.

   Note: With these Ant files, you can also:
   - Uninstall a MobileFirst Data Proxy, with the target uninstall.
   - Update a MobileFirst Data Proxy with the target minimal-update to apply a fix pack.
5. Save the Ant file. You might need it later to apply a fix pack or perform an upgrade. For more information, see "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21. If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.

Manually installing the MobileFirst Data Proxy

You can install the MobileFirst Data Proxy manually, and configure your application server accordingly.

Configuring WebSphere Application Server Liberty profile for MobileFirst Data Proxy manually:

To configure WebSphere Application Server Liberty profile for MobileFirst Data Proxy manually, you must modify the server.xml file. You must also install a feature in the usr/extension directory that is shared between servers, and add an environment variable in a server.env file that specifies the MobileFirst runtime, which provides the authentication service for the MobileFirst Data Proxy.

Procedure

1. Install the OAuthTai feature that implements the Trust Association Interceptor (TAI) for IBM MobileFirst Platform Foundation for iOS.
   a. Ensure that the Liberty usr directory (see note) contains a subdirectory extension/lib. If this subdirectory does not exist, you must create it.
   b. Copy the file *product_install_dir*/WorklightServer/external-server-libraries/com.ibm.worklight.oauth.tai_1.0.0.jar to usr/extension/lib/.
   c. Copy the file *product_install_dir*/WorklightServer/external-server-libraries/OAuthTai-1.0.mf to usr/extension/lib/.

   Note:

   Where *usr* is the usr directory for WebSphere Application Server Liberty profile. For more information, see the Directory locations and properties page in the WebSphere Application Server Liberty Core user documentation. It is typically in *liberty_install_dir*/usr but its location can be redefined with a variable in *liberty_install_dir*/etc/server.env.

   product_install_dir is the installation directory for MobileFirst Server.
2. Install the MobileFirst Data Proxy WAR file.
   a. Copy the following WAR file to the apps directory of the Liberty server: product_install_dir/Datastore/imf-data-proxy.war.

   Note: the apps directory is in the same directory as the server.xml file.
3. Edit the server.xml file.
   a. Ensure that the <featureManager> element contains at least the following <feature> elements:

```
<feature>jaxrs-1.1</feature>
<feature>restConnector-1.0</feature>
<feature>jndi-1.0</feature>
<feature>appSecurity-2.0</feature>
<feature>usr:OAuthTai-1.0</feature>
```

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

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

   c. Configure the Trust Association Interceptor:

```
<usr_OAuthTAI id="myOAuthTAI" cacheSize="1000">
  <securityConstraint securedURLs="/datastore/*"
                      scope="cloudant"
                      httpMethods="All"/>
</usr_OAuthTAI>
```

   d. If your server is not configured with a **basicRegistry** or an **ldapRegistry**, add an empty **basicRegistry**:

```
<basicRegistry> </basicRegistry>
```

   Note: There can be only one **basicRegistry** per server.xml file. You must perform this step only if there is no other **basicRegistry** or **ldapRegistry** defined in your server.

   e. Declare the MobileFirst Data Proxy application:

```
<application id="datastore" name="datastore" location="imf-data-proxy.war" type="war">
    <application-bnd>
        <security-role name="TAIUserRole">
            <special-subject type="ALL_AUTHENTICATED_USERS" />
        </security-role>
    </application-bnd>
    <classloader delegation="parentLast">
        <commonLibrary id="worklightlib_datastore">
            <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.
        </commonLibrary>
    </classloader>
</application>

<!-- Declare the JNDI properties for the MobileFirst Data Proxy. -->
<jndiEntry jndiName="datastore/CloudantProxyDbAccount" value='"hostname"'/>
<jndiEntry jndiName="datastore/CloudantProtocol" value='"http"'/>
<jndiEntry jndiName="datastore/CloudantPort" value='"80"'/>
<jndiEntry jndiName="datastore/CloudantProxyDbAccountUser" value='"cloudantuser"'/>
<jndiEntry jndiName="datastore/CloudantProxyDbAccountPassword" value='"cloudantpassword"'/>
```

Where:

- `datastore` is the context root for the application. `datastore` is used in the application name and ID, and as a prefix for the `jndiName` values, and in the `securityConstraint` of the MobileFirst Data Proxy.
- The values for the following parameters correspond to:
  - `CloudantProxyDbAccount` is the Cloudant database account or host name of the Cloudant Local server.
  - `CloudantProtocol` is the protocol to connect to the Cloudant HAProxy (http or https).
  - `CloudantPort` is the port number to connect to the Cloudant HAProxy.
  - `CloudantProxyDbAccountUser` is the login of the Cloudant user.
  - `CloudantProxyDbAccountPassword` is the password of the Cloudant user. This password can be encrypted with the `securityUtility` feature of WebSphere Application Server Liberty profile.

4. Edit or create a file `server.env` in the Liberty server directory, and add this content:

   ```
   publicKeyServerUrl=http://hostname:9080/worklight
   ```

   Where the value of the **`publicKeyServerUrl`** environment variable is the URL to the MobileFirst runtime that runs the mobile apps which use the MobileFirst Data Proxy.

Configuring WebSphere Application Server full profile and WebSphere Application Server Network Deployment for MobileFirst Data Proxy manually:

To configure WebSphere Application Server full profile or WebSphere Application Server Network Deployment for MobileFirst Data Proxy manually, you must follow these steps.

About this task

First, you must select a context root for the MobileFirst Data Proxy application. In the following topics, the context root is referred to as `/datastore`.

Then you must install the MobileFirst OAuth Trust Association Interceptor (TAI). To install this component, you have two choices, which are described in the section "Installing the MobileFirst OAuth Trust Association Interceptor (TAI)" on page 6-205.

Finally, you must install the MobileFirst Data Proxy application, as described in the section "Installing the MobileFirst Data Proxy application" on page 6-207.

Installing the MobileFirst OAuth Trust Association Interceptor (TAI):

It is mandatory to install the MobileFirst OAuth Trust Association Interceptor to run the MobileFirst Data Proxy. You can either install the TAI on the default security domain of a cell, or on a specific security domain.

About this task

There are two options to install MobileFirst OAuth Trust Association Interceptor. Review the following topics to learn about the procedures for each option.

Installing the MobileFirst OAuth Trust Association Interceptor on the default security domain:

You can choose to install the TAI on the default security domain of a cell. This is convenient for a standalone web server, but on WebSphere Application Server Network Deployment, this means that the TAI is active on all servers and clusters of the cell. It must also be installed, and maintained for fix packs, on all nodes of the cell.

Procedure

1. Copy the file *product_install_dir*/WorklightServer/external-server-libraries/com.ibm.worklight.oauth.tai_1.0.0.jar to ${WAS_INSTALL_ROOT}/lib/ext.

   Note: On WebSphere Application Server Network Deployment, you must perform this operation on every node of the WebSphere Application Server cell.
2. Create a configuration file with this content:
   ```
   <?xml version="1.0" encoding="UTF-8"?>

     <OAuthTAI >
       <!-- Security constraint. -->
       <securityConstraint securedURLs="/datastore/*" scope="cloudant" httpMethods="All"/>
     </OAuthTAI>
   ```
   Where datastore is the context root of the application, as defined in the section *About this task* of the topic "Configuring WebSphere Application Server full profile and WebSphere Application Server Network Deployment for MobileFirst Data Proxy manually" on page 6-204.
3. Copy this configuration file in the config directory of the WebSphere Application Server profile, or the WebSphere Application Server Deployment Manager profile for WebSphere Application Server Network Deployment (this directory is synchronized with the nodes). For example:

   ${*USER_INSTALL_ROOT*}/config/cells/<*cellName*>/com.worklight.oauth.tai.OAuthTAI.conf

   Where *USER_INSTALL_ROOT* is the profile directory, and <cellName> must be replaced by the actual name of the WebSphere Application Server cell. The directory should already exist.
4. Open the WebSphere Application Server Console.
5. Go to **Security** > **Global Security** > **Authentication**, and select **Enable LTPA**.
6. Go to **Security** > **Global Security** > **Web and SIP security** > **Trust association**.
   a. Enable trust association.
   b. In the **Interceptors** tab, create a new interceptor by clicking **New**.

   c. Set the following settings for this interceptor:
      - Interceptor class name: `com.worklight.oauth.tai.OAuthTAI`.
      - Custom properties:
        - Name: `configFileLocation`
        - Value: `${`*`USER_INSTALL_ROOT`*`}/config/cells/<`*`cellName`*`>/com.worklight.oauth.tai.OAuthTAI.conf`

      Note: Keep `${`*`USER_INSTALL_ROOT`*`}` as a variable, especially if you install on WebSphere Application Server Network Deployment. The `${`*`USER_INSTALL_ROOT`*`}` variable is defined in WebSphere Application Server.

      You must replace <*cellName*> by the actual cell name.

7. From the WebSphere Application Server console, define an environment variable for each server of the cell that points to the MobileFirst project runtime which provides the authentication service.
   a. From the WebSphere Application Server Console, go to **Servers** > **Server Types** > **WebSphere application servers** > *your_server* > **Java and Process Management** > **Process Definition** > **Environment Entries** > **New**.
   b. Create an environment with the following settings:

      `publicKeyServerUrl=url_to_mfp_server`

      Where `url_to_mfp_server` is the URL to the MobileFirst project runtime that provides the public key for decrypting MFP OAuth tokens. For example, it can be `http://localhost:9080/worklight` if a project runtime is installed on the same server, the port is 9080, and the context root is `/worklight`.

      The TAI will not be active before you restart the application server. For WebSphere Application Server Network Deployment, the nodes must also be synchronized.

What to do next

Follow the steps in “Installing the MobileFirst Data Proxy application” on page 6-207

Installing the MobileFirst OAuth Trust Association Interceptor on a specific security domain:

You can choose to install the TAI on a specific security domain. In this case, only servers or clusters that use that security domain can run the MobileFirst Data Proxy.

Procedure

1. Open the WebSphere Application Server Console.
   a. Create a security domain by clicking **Security** > **Global Security** > **Security domains** > **New**.
   b. Configure the security domain as follows:
      - Name: `MobileFirstOAuthDomain`
      - Description: Security Domain with MobileFirst Platform Foundation TAI Enabled
   c. Click the security domain that you created.
   d. Go to **Security Attributes** > **Trust Association**, and select **Enable Trust Association**.
   e. Go to **Security Attributes** > **Trust Association** > **Interceptors** and set the properties as follows:

- Interceptor class name: `com.worklight.oauth.tai.OAuthTAI`.
- Custom properties:
  - Name: `configFileLocation`
  - Value: `${`*`USER_INSTALL_ROOT`*`}/config/cells/<`*`cellName`*`>/com.worklight.oauth.tai.OAuthTAI.conf`

  Note: Keep `${`*`USER_INSTALL_ROOT`*`}` as a variable, especially if you install on WebSphere Application Server Network Deployment. The `${`*`USER_INSTALL_ROOT`*`}` variable is defined in WebSphere Application Server.

  You must replace <*cellName*> by the actual cell name.

f. In **Assigned Scopes**, assign the security domain to the server or cluster that runs the MobileFirst Data Proxy.

2. Copy the file *`product_install_dir`*`/WorklightServer/external-server-libraries/com.ibm.worklight.oauth.tai_1.0.0.jar` to `${WAS_INSTALL_ROOT}/lib/ext`.

   Note: You must perform this operation on every node that hosts a server with the `MobileFirstOAuthDomain` security domain.

3. Create the TAI configuration file and copy it in the `config` directory of the WebSphere Application Server profile. For more information, see steps 1 on page 6-205 and 2 on page 6-205 of “Installing the MobileFirst OAuth Trust Association Interceptor on the default security domain” on page 6-205.
4. From the WebSphere Application Server console, define an environment variable for each server with the `MobileFirstOAuthDomain` security domain.
   a. Go to **Servers** > **Server Types** > **WebSphere application servers** > *your_server* > **Java and Process Management** > **Process Definition** > **Environment Entries** > **New**.
   b. Create an environment with the following settings:

   ```
   publicKeyServerUrl=url_to_mfp_server
   ```

   Where `url_to_mfp_server` is the URL to the MobileFirst project runtime that provides the public key for decrypting MFP OAuth tokens. For example, it can be `http://localhost:9080/worklight` if a project runtime is installed on the same server, the port is 9080, and the context root is `/worklight`.

   The TAI will not be active before you restart the application server. For WebSphere Application Server Network Deployment, the nodes must also be synchronized.

What to do next

Follow the steps in “Installing the MobileFirst Data Proxy application”

Installing the MobileFirst Data Proxy application:

When the previous configuration steps of your application server are done, you must install, then start the MobileFirst Data Proxy application.

Procedure

1. 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. Click **OK**.

d. Save the changes.

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

2. Review the server class loader policy:

a. Click **Servers** > **Server Types** > **WebSphere application servers**.

b. Select the server that is used for the MobileFirst Data Proxy.

- 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 MobileFirst applications to **parent-first**.

3. 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 MobileFirst Server installation directory: `*product_install_dir*/Datastore`.

c. Select `imf-data-proxy.war`, and 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 `/datastore`. This is defined in the section *About this task* of the topic "Configuring WebSphere Application Server full profile and WebSphere Application Server Network Deployment for MobileFirst Data Proxy manually" on page 6-204

h. Click **Next**.

i. In **Map environment entries for web modules**, enter the following values:

- for the `CloudantProxyDbAccount` entry, enter the host name of the Cloudant database.
- for the `CloudantProtocol` entry, enter the protocol used to connect to Cloudant. The possible values are `http` or `https`.
- for the `CloudantPort` entry, enter the port of the Cloudant database, which is by default 80 for `http` and 443 for `https`.
- for the `CloudantProxyDbAccountUser` entry, enter the Cloudant user that can log to Cloudant.
- for the `CloudantProxyDbAccountPassword` entry, enter the Cloudant user's password.

j. Click **Next**.

k. In **Map security roles to users or groups**, select **TAIUserRole**.

l. Select **Map Special Subjects** > **All Authenticated Users in Application's Realm**.

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

n. Click **Save**.

4. 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 `imf-data-proxy_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 **imfdata** 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 `imf-data-proxy_war` and click **Start**.

Configuring the application server to access the Cloudant database through HTTPS

Whether you installed the MobileFirst Data Proxy manually or with Ant tasks, if you access the Cloudant database through HTTPS, and your application server is WebSphere Application Server full profile, you must configure your certificates. If your application server is WebSphere Application Server Liberty profile, and a self-signed certificate is used to access the Cloudant database, an extra configuration step is required.

Procedure

For WebSphere Application Server full profile, you must import the Cloudant signer certificate in the WebSphere Application Server truststore to access the Cloudant database through HTTPS. If you connect to Cloudant through the HTTPS protocol, follow steps 1 to 7.

1. Open the WebSphere Application Server console.
2. Go to **Security** > **SSL Certificates and Key Management**.
3. In **Related Items**, click **Key stores and certificates**.
4. Select **NodeDefaultTrustStore**.
5. Select **Additional Properties** > **Signer certificates**.
6. Click **Retrieve from port**.

   a. Enter the Cloudant host name and the port, which is by default 443.

   b. Select an alias, for example `Cloudant trust store`.

   c. Click **Retrieve signer information**.

   d. Click **OK**.

7. Click **Save**.

For WebSphere Application Server Liberty profile, if you access the Cloudant database through HTTPS with a self-signed certificate, you must import this certificate in the cacerts truststore of the JVM that is used by your Liberty server, which you find in: `JAVA_INSTALL_DIR\jre\lib\security\cacerts`.

8. Use the **keytool** command that is available in both IBM JRE and Oracle JRE, as of Java 6.
9. For more information, see the Keytool section of the IBM SDK, Java Technology Edition user documentation.

   Note: The password to access this truststore is `changeit`.

Configuring the connection between the MobileFirst Data Proxy and the Analytics server

If you installed MobileFirst Operational Analytics on a server, you can optionally configure the MobileFirst Data Proxy so that it can send events to this server.

To configure the connection between the MobileFirst Data Proxy and the Analytics server, you must follow the instructions from the section *Reporting analytics*, in the page "Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty" on page 8-199.

Installing and configuring the Application Center

You install the Application Center as part of the MobileFirst Server installation.

The Application Center is part of MobileFirst Server. You can install the Application Center with one of the following methods:

- Installation with IBM Installation Manager
- Installation with Ant tasks
- Manual installation

Optionally, you can create the database of your choice before you install MobileFirst Server with the Application Center.

After you installed the Application Center in the web application server of your choice, you have additional configuration to do. For more information, see "Configuring Application Center after installation" on page 6-243.

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 MobileFirst Server" on page 6-42.

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 6-5.

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 6-32.
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 MobileFirst 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 MobileFirst Server installer.

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

Creating the DB2 database for Application Center:

During IBM MobileFirst Platform Foundation for iOS 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 instances of IBM MobileFirst Platform 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 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 MobileFirst 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 MobileFirst 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 MobileFirst Platform Foundation for iOS runs.

Creating the Oracle database for Application Center:

During the installation, the installer can create the Application Center database, except for the Oracle 12c database type, or the user and schema inside an existing database for you.

About this task

The installer can create the database, except for the Oracle 12c database type, or the user and schema inside an existing database 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 instances of IBM MobileFirst Platform 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 tablespace: **`USERS`**
        - Temporary tablespace: **`TEMP`**
        - Status: **`Unlocked`**
        - Add system privilege: **`CREATE SESSION`**
        - Add system privilege: **`CREATE SEQUENCE`**
        - Add system privilege: **`CREATE TABLE`**
        - Add quota: **`Unlimited for tablespace USERS`**
   - 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 DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USE
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO APPCENTER;
DISCONNECT;
```

Installing Application Center in WebSphere Application Server Network Deployment

To install Application Center 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 MobileFirst Platform Foundation for iOS 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 Application Center to be installed. The following table lists the available scopes:

Table 6-51. Selecting the required scope.

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

4. Restart the target servers by following the procedure in "Completing the installation" on page 6-215.

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 Application Center in different configurations or even different versions of Application Center, 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 IBM MobileFirst Platform Application Center or MobileFirst Server.

  The MobileFirst 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 MobileFirst Server completed.
- When you are using WebSphere Application Server Liberty or Apache Tomcat.
- After you upgraded from a previous version of MobileFirst 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 MobileFirst 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 MobileFirst Operations Console is not installed by default. To install a MobileFirst Operations Console, you need to follow the steps in "Deploying MobileFirst projects" on page 11-1.

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 the Java EE security roles on WebSphere Application Server full profile" on page 6-244.

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 the Java EE security roles on WebSphere Application Server Liberty profile" on page 6-246.

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 the Java EE security roles on Apache Tomcat" on page 6-247.

Installing the Application Center with Ant tasks

Learn about the Ant tasks that you can use to install Application Center.

Creating and configuring the database for Application Center with Ant tasks

If you did not manually create the database, you can use Ant tasks to create and configure your database for Application Center. If your database already exists, you can perform only the configuration steps with Ant tasks.

Before you begin

Make sure that a database management system (DBMS) is installed and running on a database server, which can be on the same computer, or a different one.

The Ant tasks for Application Center are in the ApplicationCenter/configuration-samples directory of the MobileFirst Server distribution.

If you want to start the Ant task from a computer where MobileFirst Server is not installed, you must copy the following files to that computer:

- The library *mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar
- The directory that contains binary files of the aapt program, from the Android SDK platform-tools package: *mf_server_install_dir*/ApplicationCenter/tools/android-sdk
- The Ant sample files that are in *mf_server_install_dir*/ApplicationCenter/configuration-samples

Note: The *mf_server_install_dir* placeholder represents the directory where you installed MobileFirst Server.

About this task

- If you did not create your database manually, as described in "Optional creation of databases" on page 6-211, follow steps 1 to 3 on page 6-217.
- If your database already exists, you must create only the database tables. Follow steps 4 on page 6-217 to 7 on page 6-217.

Procedure

If you did not create your database manually, as described in "Optional creation of databases" on page 6-211, complete the following steps:

1. Copy the sample Ant file that corresponds to your DBMS. The files for creating a database are named after the following pattern:

```
create-appcenter-database-<dbms>.xml
```

2. Edit the Ant file, and replace the placeholder values with the properties at the beginning of the file.
3. Run the following commands to create the Application Center database:

   ```
   ant -f create-appcenter-database-<dbms>.xml databases
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.

If the database already exists, then you must create only the database tables by completing the following steps:

4. Copy the sample Ant file that corresponds to both your application server, and your DBMS. The files for configuring an existing database are named after this pattern:

   ```
   configure-appcenter-<appServer>-<dbms>.xml
   ```

5. Edit the Ant file, and replace the placeholder values with the properties at the beginning of the file.
6. Run the following commands to configure the database:

   ```
   ant -f configure-appcenter-<appServer>-<dbms>.xml databases
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.
7. Save the Ant file. You might need it later to apply a fix pack, or perform an upgrade.

   For more information, see "Deploying the Application Center console and services with Ant tasks."

   If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.

What to do next

Follow the procedure at "Deploying the Application Center console and services with Ant tasks."

See also:
- "Ant tasks for installation of Application Center" on page 15-45
- "Sample configuration files" on page 15-66

Deploying the Application Center console and services with Ant tasks

Use Ant tasks to deploy the Application Center Console and Services to an application server, and configure data sources, properties, and database drivers that are used by Application Center.

Before you begin

- Complete the procedure at "Creating and configuring the database for Application Center with Ant tasks" on page 6-216.
- You must run the Ant task on the computer where the application server is installed, or the Network Deployment Manager for WebSphere Application Server Network Deployment. If you want to start the Ant task from a computer where MobileFirst Server is not installed, you must copy the following files and directories to that computer:
  - The library *mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar

- The web applications (WAR and EAR files) in *mf_server_install_dir*/ApplicationCenter/console
- The directory that contains the binary files of the aapt program, from the Android SDK platform-tools package: *mf_server_install_dir*/ApplicationCenter/tools/android-sdk
- The Ant sample files that are in *mf_server_install_dir*/ApplicationCenter/configuration-samples

Note: The *mf_server_install_dir* placeholder represents the directory where you installed MobileFirst Server.

Procedure

1. Copy the Ant file that corresponds both to your application server, and your DBMS. The files for configuring Application Center are named after the following pattern:

   configure-appcenter-<*appserver*>-<*dbms*>.xml
2. Edit the Ant file, and replace the placeholder values with the properties at the beginning of the file.
3. Run the following command to deploy the Application Center Console and Services to an application server:

   ```
   ant -f configure-appcenter-<appserver>-<dbms>.xml install
   ```

   You can find the Ant command in *mf_server_install_dir*/shortcuts.

   Note: With these Ant files, you can also do the following actions:
   - Uninstall Application Center, with the target uninstall.
   - Update Application Center with the target minimal-update, to apply a fix pack.
4. Save the Ant file. You might need it later to apply a fix pack or perform an upgrade. If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.
5. If you installed on WebSphere Application Server Liberty profile, or Apache Tomcat, check that the aapt program is executable for all users. If needed, you must set the proper user rights. For example, on UNIX/Linux systems:

   ```
   $ chmod a+x mf_server_install_dir/ApplicationCenter/tools/android-sdk/*/aapt*
   ```

Manually installing Application Center

A reconfiguration is necessary for the MobileFirst 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.

On application servers other than Apache Tomcat, you can deploy Application Center from two WAR files or one EAR file.

Restriction: Whether you install Application Center with IBM Installation Manager as part of the MobileFirst Server installation or manually, remember that "rolling updates" 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. See "In-place upgrade or rolling upgrade to MobileFirst Server V7.1.0" on page 7-28.

Configuring the DB2 database manually for IBM MobileFirst Platform 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 6-211.
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 product_install_dir/ApplicationCenter/databases/create-appcenter-db2.sql -t
   ```

Configuring Liberty profile for DB2 manually for Application Center:

You can set up and configure your DB2 database manually for Application Center with WebSphere Application Server Liberty profile.

Before you begin

Complete the DB2 Database Setup procedure before continuing.

Procedure

1. Add the DB2 JDBC driver JAR file to *$LIBERTY_HOME*/wlp/usr/shared/resources/db2.

   If that directory does not exist, create it. You can retrieve the file in one of two ways:

   - Download it from DB2 JDBC Driver Versions.
   - Fetch it from the *db2_install_dir*/java on the DB2 server directory.
2. Configure the data source in the *$LIBERTY_HOME*/wlp/usr/servers/worklightServer/server.xml file as follows:

   In this path, you can replace worklightServer by the name of your server.

   ```
   <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>
   ```

   The worklight placeholder after **user=** is the name of the system user with CONNECT access to the **APPCNTR** database that you have previously created. The worklight placeholder 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 computer).

   DB2 has a user name and password length limit of 8 characters for UNIX and Linux systems, and 30 characters for Windows.
3. You can encrypt the database password with the securityUtility program in <liberty_install_dir>/bin.

Configuring WebSphere Application Server for DB2 manually for Application Center:

You can set up and configure your DB2 database manually for Application Center with WebSphere Application Server.

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 stand-alone 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 and its associated license files, if any, to the directory that you determined in step 1.

   You can retrieve the driver file in one of two ways:

   - Download it from DB2 JDBC Driver Versions.
   - Fetch it from the *db2_install_dir*/java directory on the DB2 server.

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, 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 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 6-221 to 4h on page 6-221.

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

Before you contiue, complete the DB2 database setup procedure.

Procedure

1. Add the DB2 JDBC driver JAR file.

   You can retrieve this JAR file in one of the following ways:

   - 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. Prepare an XML statement that defines the data source, as shown in the following code example.

```
<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;"/>
```

   The **`worklight`** parameter after **`username=`** is the name of the system user with "CONNECT" access to the **`APPCNTR`** database that you have previously created. The **`password`** parameter 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 enforces limits on the length of user names and passwords.

   - For UNIX and Linux systems: 8 characters
   - For Windows: 30 characters

3. Insert this statement in the `server.xml` file, as indicated in "Configuring Apache Tomcat for Application Center manually" on page 6-238.

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 6-224
   - "Configuring Apache Tomcat for Derby manually for Application Center" on page 6-226

Setting up your Apache Derby database manually for Application Center:

You can set up your Apache Derby database for Application Center manually.

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.

   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.

   For supported versions of Apache Derby, see "System requirements" on page 2-9.
   The script displays `ij` version number.
2. At the command prompt, enter the following commands:
   ```
   connect 'jdbc:derby:APPCNTR;user=APPCENTER;create=true';
   run '<product_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.

Before you begin

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.EmbeddedConnectionPoolDataSou
  <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:

You can set up and configure your Apache Derby database manually for Application Center with WebSphere Application Server.

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.

   If this directory does not exist, create it.

   - 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`.
2. Add the `Derby JAR` file from `*product_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** data source 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 6-223.

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** data source that you created.

x. Optional: Only if you are not on the console of a WebSphere Application Server Deployment Manager, click **test connection**.

Configuring Apache Tomcat for Derby manually for Application Center:

You can set up and configure your Apache Derby database manually for Application Center with the Apache Tomcat application server.

About this task

Complete the Apache Derby database setup procedure before continuing.

Procedure

1. Add the Derby JAR file from *product_install_dir*/ApplicationCenter/tools/lib/derby.jar to the directory $TOMCAT_HOME/lib.
2. Prepare an XML statement that defines the data source, as shown in the following code example.

   ```
   <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"/>
   ```
3. Insert this statement in the server.xml file, as indicated in "Configuring Apache Tomcat for Application Center manually" on page 6-238.

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 6-212.
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 product_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 MobileFirst Platform Foundation for iOS 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.
3. Add the following property to your MySQL option file: `innodb_log_file_size = 250M`

   For more information about the **innodb_log_file_size** property, see the MySQL documentation, section innodb_log_file_size.

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.

Before you begin

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. You can 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).
3. You can encrypt the database password with the securityUtility program in `<liberty_install_dir>/bin`.

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 Serverr 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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the `server.xml` file, as indicated in “Configuring Apache Tomcat for Application Center manually” on page 6-238.

```
<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"/>
```

Configuring the Oracle database manually for IBM MobileFirst Platform 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 6-212.
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 computer, make sure that the database is started next time the Oracle installation is restarted. To this effect, make sure that 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.
   - To create the user for the Application Center database/schema, by using Oracle Database Control, proceed as follows:
     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: Unlocked
        Add system privilege: CREATE SESSION
        Add system privilege: CREATE SEQUENCE
        Add system privilege: CREATE TABLE
        Add quota: Unlimited for tablespace USERS
        ```
   - To create the user by using Oracle SQLPlus, enter the following commands:
     ```
     CONNECT SYSTEM/<SYSTEM_password>@ORCL
     CREATE USER APPCENTER IDENTIFIED BY password DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS;
     GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO APPCENTER;
     DISCONNECT;
     ```

3. Create the tables for the Application Center database:
   a. Using the Oracle SQLPlus command-line interpreter, create the tables for the Application Center database by running the `create-appcenter-oracle.sql` file:

      ```
      CONNECT APPCENTER/APPCENTER_password@ORCL
      @product_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:

You can set up and configure your Oracle database manually for Application Center with WebSphere Application Server Liberty profile by adding the JAR file of the Oracle JDBC driver.

Before you begin

Before continuing, set up the Oracle database.

Procedure

1. Add the JAR file of the Oracle JDBC driver 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/mobileFirstServer/server.xml` file as shown in the following JNDI code example:

   Note: In this path, you can replace `mobileFirstServer` with the name of your server.

   ```
   <!-- 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).

   Note: For more information on how to connect the Liberty server to the Oracle database with a service name, or with a URL, see the WebSphere Application Server Liberty Core 8.5.5 documentation, section **properties.oracle**.

3. You can encrypt the database password with the securityUtility program in `<liberty_install_dir>/bin`.

What to do next

For more steps to configure Application Center, see "Deploying the Application Center WAR files and configuring the application server manually" on page 6-234.

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 6-52. 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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the server.xml file, as indicated in “Configuring Apache Tomcat for Application Center manually” on page 6-238

```
<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"/>
```

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.

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

Note: Using the MobileFirst 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 *product_install_dir*/`ApplicationCenter/console`.

Configuring the Liberty profile for Application Center manually:

After you deploy WAR files, to configure WebSphere Application Server Liberty profile manually for Application Center, 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 6-218, you must make the following modifications to the `server.xml` file.

Procedure

1. Ensure that the `<featureManager>` element contains at least the following `<feature>` elements:
   - For WebSphere Application Server Liberty profile V8.5.0.x:
     ```
     <feature>ssl-1.0</feature>
     <feature>servlet-3.0</feature>
     <feature>jdbc-4.0</feature>
     <feature>appSecurity-1.0</feature>
     <feature>jndi-1.0</feature>
     ```
   - For WebSphere Application Server Liberty profile V8.5.5.0 and later:
     ```
     <feature>jdbc-4.0</feature>
     <feature>appSecurity-2.0</feature>
     <feature>servlet-3.0</feature>
     ```
2. Add the following declarations for Application Center:

```
<!-- Declare the 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>
  <classloader delegation="parentLast">
    <commonLibrary>
      <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.jar"/>
    </commonLibrary>
  </classloader>
</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>
```

The groups and users that are defined in the `basicRegistry` element are example logins which you can use to test Application Center. Similarly, the groups that are defined in the `<security-role name="appcenteradmin">` for the Application Center console and the Application Center service are examples. For more information about how to modify these groups, see “Configuring the Java EE security roles on WebSphere Application Server Liberty profile” on page 6-246.

3. If the database is Oracle, add the **`commonLibraryRef`** attribute to the class loader of the Application Center service 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. This ID is declared in the procedure that is documented in “Configuring Liberty profile for Oracle manually for Application Center” on page 6-231.

4. Copy the Application Center WAR files to your Liberty server.
   - On UNIX and Linux systems:

```
mkdir -p $LIBERTY_HOME/wlp/usr/servers/<server_name>/apps
 cp product_install_dir/ApplicationCenter/console/*.war
```

- On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\servers\<server_name>\apps
    copy /B product_install_dir\ApplicationCenter\console\appcenterconsole.war LIBERTY_HOME\
wlp\usr\servers\<server_name>\apps\appcenterconsole.war
    copy /B product_install_dir\ApplicationCenter\console\applicationcenter.war LIBERTY_HOME\
wlp\usr\servers\<server_name>\apps\applicationcenter.war
```

5. Start the Liberty server.

What to do next

For more steps to configure Application Center, see “Configuring the Java EE security roles on WebSphere Application Server Liberty profile” on page 6-246.

Configuring WebSphere Application Server for Application Center manually:

After you deploy WAR files, to configure WebSphere Application Server for Application Center manually, you must configure variables, custom properties, and class loading policies.

Before you begin

Make sure that a WebSphere Application Server profile exists.

Procedure

1. Log on to the WebSphere Application Server administration console for your IBM MobileFirst Platform 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.
3. Create the Application Center JDBC data source and provider.

   See the appropriate section in “Manually installing Application Center” on page 6-218.
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. Go to the MobileFirst Server installation directory *mfserver_install_dir*/`ApplicationCenter/console`.
   c. Select **appcenterconsole.war** and 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 `/appcenterconsole`.

h. Click **Next** until you reach the "Map security roles to users or groups" page.

i. Select all roles, click **Map Special Subjects** and select **All Authenticated in Application's Realm**.

j. Click **Next** until you reach the Summary page.

k. Click **Finish** and save the configuration.

5. Configure the class loader policies and then start the application:
   a. Click **Applications** > **Application types** > **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 **ApplicationCenterConsole**.
   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 **appcenterconsole_war** and click **Start**.
6. Install the WAR file for Application Center services.
   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 MobileFirst Server installation directory *`mfserver_install_dir`*`/ApplicationCenter/console`.
   c. Select **applicationcenter.war** and 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.
   g. Click **Browser** and select the data source with the `jdbc/AppCenterDS` JNDI name.
   h. Click **Apply**.
   i. In the **Context Root** field, type `/applicationcenter`.
   j. Click **Next** until you reach the "Map security roles to users or groups" page.
   k. Select all roles, click **Map Special Subjects**, and select **All Authenticated in Application's Realm**.
   l. Click **Next** until you reach the Summary page.
   m. Click **Finish** and save the configuration.
7. Repeat step 5.
   a. Select **applicationcenter.war** from the list of applications in substeps b and k.
   b. Select **ApplicationCenterServices** in substep g.

8. Review the server class loader policy: Depending on your version of WebSphere Application Server, click **Servers** > **Server Types** > **Application Servers** or **Servers** > **Server Types** > **WebSphere application servers** and then select the server.
   - If the class loader policy is set to **Multiple**, do nothing.
   - If the class loader policy is set to **Single** and **Class loading mode** is set to **Classes loaded with local class loader first (parent last)**, do nothing.
   - If **Classloader policy** is set to **Single** and **Class loading mode** is set to **Classes loaded with parent class loader first**, set **Classloader policy** to **Multiple** and set the class loader policy of all applications other than MobileFirst applications to **Classes loaded with parent class loader first**.
9. Save the configuration.

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 (by default `9080`).

What to do next

For more steps to configure the Application Center, see "Configuring the Java EE security roles on WebSphere Application Server full profile" on page 6-244.

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 6-218.
2. Edit *`tomcat_install_dir`*`/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 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 host name 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 6-222
- "Configuring Apache Tomcat for Derby manually for Application Center" on page 6-226
- "Configuring Apache Tomcat for MySQL manually for Application Center" on page 6-229
- "Configuring Apache Tomcat for Oracle manually for Application Center" on page 6-233

3. Copy the Application Center WAR files to Tomcat.
   - On UNIX and Linux systems: `cp` *`product_install_dir`*`/ApplicationCenter/console/*.war TOMCAT_HOME/webapps`
   - On Windows systems:

     ```
     copy /B product_install_dir\ApplicationCenter\console\appcenterconsole.war tomcat_install_di
     copy /B product_install_dir\ApplicationCenter\console\applicationcenter.war tomcat_install_a
     ```
4. Start Tomcat.

What to do next

For more steps to configure the Application Center, see "Configuring the Java EE security roles on Apache Tomcat" on page 6-247.

Deploying the Application Center EAR file and configuring the application server manually

As an alternative to the MobileFirst Server installer procedure, you can use a manual procedure to deploy the Application Center EAR file and configure your WebSphere application server manually.

Before you begin

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

About this task

The procedure to deploy the Application Center EAR file manually to an application server depends on the type of application server. Manual deployment is supported only for WebSphere Application Server Liberty profile and WebSphere Application Server.

Tip: It is more reliable to install Application Center through the MobileFirst Server installer than manually. Therefore, whenever possible, use the MobileFirst Server installer. If, however, you prefer the manual procedure, deploy the `appcentercenter.ear` file, which you can find in the *`product_install_dir`*`/ApplicationCenter/console` directory.

Configuring the Liberty profile for Application Center manually:

After you deploy the Application Center EAR file, to configure WebSphere Application Server Liberty profile manually for Application Center, 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, you must make the following modifications to the `server.xml` file.

Procedure

1. Ensure that the `<featureManager>` element contains at least the following `<feature>` elements:
   -

     For WebSphere Application Server Liberty profile V8.5.0.x:
     ```
     <feature>ssl-1.0</feature>
     <feature>servlet-3.0</feature>
     <feature>jdbc-4.0</feature>
     <feature>appSecurity-1.0</feature>
     <feature>jndi-1.0</feature>
     ```
   - For WebSphere Application Server Liberty profile V8.5.5.0 and later:
     ```
     <feature>jdbc-4.0</feature>
     <feature>appSecurity-2.0</feature>
     <feature>servlet-3.0</feature>
     ```
2. Add the following declarations for Application Center:
   ```
   <!-- The directory with binaries of the 'aapt' program, from the Android SDK's platform-tools pac
   <jndiEntry jndiName="android.aapt.dir" value="product_install_dir/ApplicationCenter/tools/android

   <!-- Declare the IBM Application Center application. -->
   <application id="applicationcenter"
                name="applicationcenter"
                location="applicationcenter.ear"
                type="ear">
     <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"/>
    </commonLi
  </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>
```

The groups and users that are defined in the `basicRegistry` element are example logins, which you can use to test Application Center. Similarly, the groups that are defined in the `<security-role name="appcenteradmin">` element are examples. For more information about how to modify these groups, see "Configuring the Java EE security roles on WebSphere Application Server Liberty profile" on page 6-246.

3. If the database is Oracle, add the **commonLibraryRef** attribute to the class loader of the Application Center service 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. This ID is declared in the procedure that is documented in "Configuring Liberty profile for Oracle manually for Application Center" on page 6-231.

4. Copy the Application Center EAR files to your Liberty server.
   - On UNIX and Linux systems:

```
mkdir -p $LIBERTY_HOME/wlp/usr/servers/server_name/apps
cp product_install_dir/ApplicationCenter/console/*.ear
```

   - On Windows systems:

```
mkdir LIBERTY_HOME\wlp\usr\servers\<server_name\appscopy /B
product_install_dir\ApplicationCenter\console\applicationcenter.ear
LIBERTY_HOME\wlp\usr\servers\<server_name\apps\applicationcenter.ear
```

5. Start the Liberty server.

What to do next

For more steps to configure Application Center, see "Configuring the Java EE security roles on WebSphere Application Server Liberty profile" on page 6-246.

Configuring WebSphere Application Server for Application Center manually:

After you deploy the Application Center EAR file, to configure WebSphere Application Server profile manually for Application Center, you must configure variables, custom properties, and class loader policies.

Before you begin

Make sure that a WebSphere Application Server profile exists.

Procedure

1. Log on to the WebSphere Application Server administration console for your IBM MobileFirst Platform 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.
3. Create the Application Center JDBC data source and provider.

   See the appropriate section in "Manually installing Application Center" on page 6-218.
4. Install the Application Center EAR 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 MobileFirst Server installation directory *`mfserver_install_dir`*`/ApplicationCenter/console`.
   c. Select **applicationcenter.ear**, and then click **Next**.
   d. On the How do you want to install the application? page, click **Detailed**, and then click **Next**.
   e. Click **Next** until you reach the "Map resource references to resources" page.
   f. Click **Browse** and select the data source with the `jdbc/AppCenterDS` JNDI name.
   g. Click **Next** until you reach the "Map context roots for web modules" page.
   h. In the **Context Root** fields, if they are not already set, type `/appcenterconsole` for the ApplicationCenterConsole module and type `/applicationcenter` for the ApplicationCenterServices module.
   i. Click **Next** until you reach the "Map security roles to users or groups" page.
   j. Select all roles, click **Map Special Subjects** and select **All Authenticated in Application's Realm**.
   k. Click **Next** until you reach the Summary page.
   l. Click **Finish** and save the configuration.
5. Configure the class loader policies and then start the application:
   a. Click **Applications** > **Application types** > **WebSphere Enterprise Applications**.
   b. From the list of applications, click **AppCenterEAR**.
   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 **ApplicationCenterConsole**.

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

i. Click **OK**.

j. From the list of modules, click **ApplicationCenterServices**.

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

l. Click **OK** twice.

m. Click **Save**.

n. Select **AppCenterEAR** and click **Start**.

6. Review the server class loader policy:

   Depending on your version of WebSphere Application Server, click **Servers** > **Server Types** > **Application Servers** or **Servers** > **Server Types** > **WebSphere application servers** and then select the server.

   - If the class loader policy is set to **Multiple**, do nothing.
   - If the class loader policy is set to **Single** and **Class loading mode** is set to **Classes loaded with local class loader first (parent last)**, do nothing.
   - If **Classloader policy** is set to **Single** and **Class loading mode** is set to **Classes loaded with parent class loader first**, set **Classloader policy** to **Multiple** and set the class loader policy of all applications other than MobileFirst applications to **Classes loaded with parent class loader first**.

7. Save the configuration.

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 (by default `9080`).

What to do next

For more steps to configure the Application Center, see "Configuring the Java EE security roles on WebSphere Application Server full profile" on page 6-244.

Configuring Application Center after installation

After you install Application Center in the web application server that you designated, you have additional configuration to do.

Configuring user authentication for Application Center

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

Application Center requires user authentication.

You must perform some configuration after the installer deploys Application Center web applications in the web application server.

Application Center has two Java Platform, Enterprise Edition (Java EE) security roles defined:

- The **appcenteruser** role that represents an ordinary user of 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.

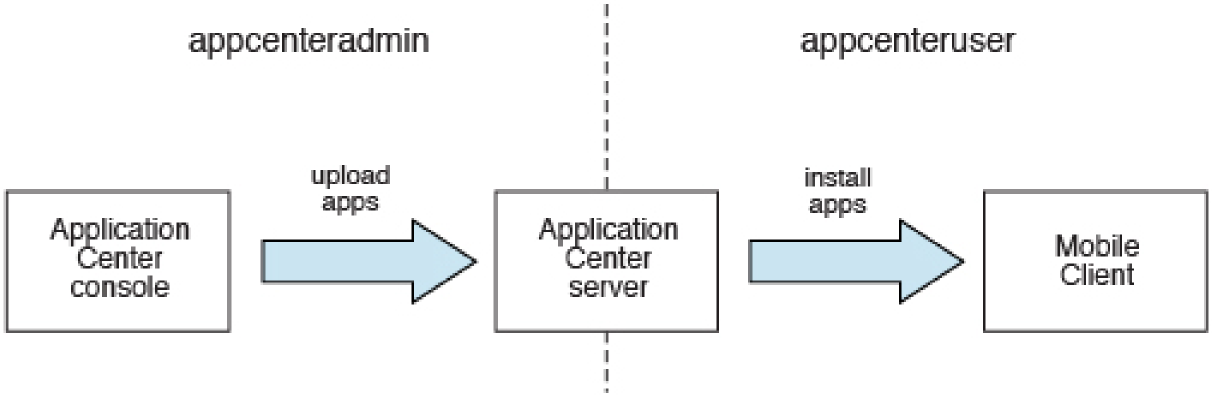


Figure 6-18. Java EE security roles of the Application Center and the components that they influence

If you choose to use an authentication method through a user repository such as LDAP, you can configure Application Center so that you can use users and groups with the user repository to define the Access Control List (ACL) of 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 6-248 for information about LDAP used with Application Center.

After you configure authentication of the users of 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 12-69 for how to build the Application Center mobile client.

Related concepts:

"Managing users with LDAP" on page 6-248
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 apps on mobile devices, you must first import the **IBMAppCenter** project into the Eclipse environment, build the project, and deploy the mobile client in the Application Center.

Configuring the Java EE security roles on WebSphere Application Server full profile:

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

Before you begin

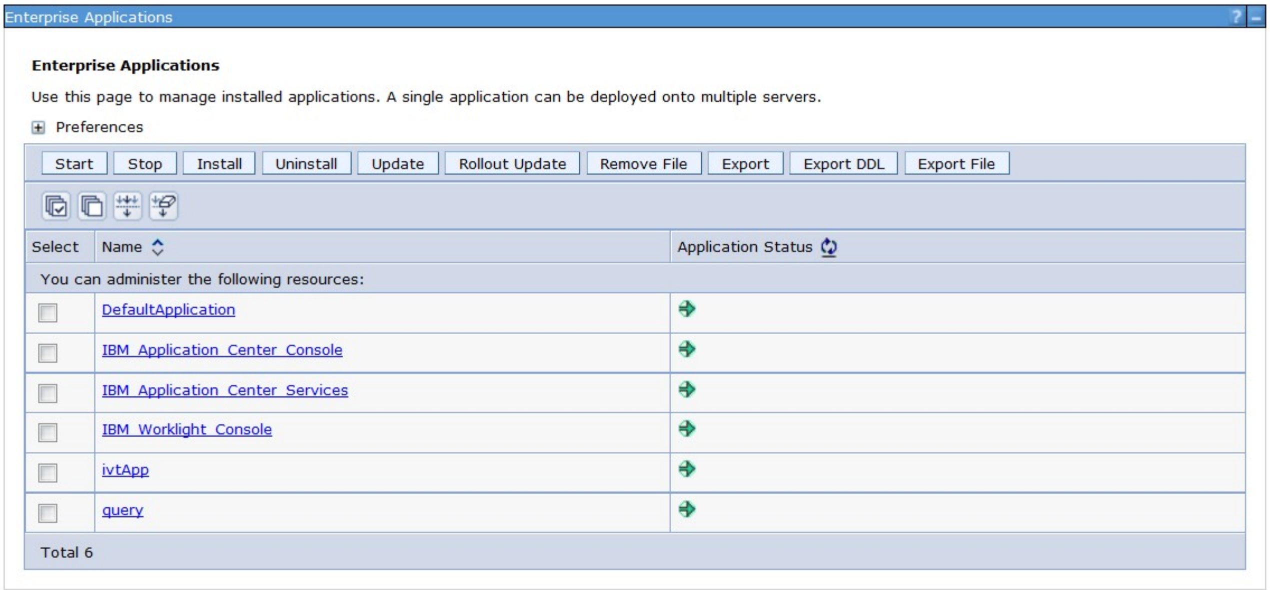
Review the definition of roles at “Configuring user authentication for Application Center” on page 6-243.

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. If you deployed WAR files, map the **`appcenteruser`** and **`appcenteradmin`** roles to a set of users as follows:
   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

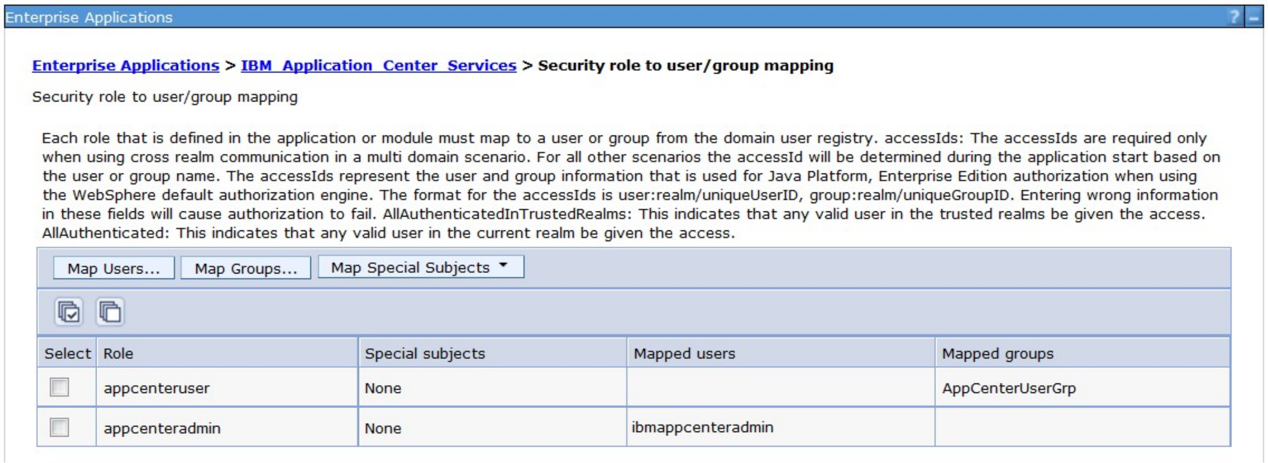
Use this page to manage installed applications. A single application can be deployed onto multiple servers.

⊞ Preferences

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 6-19. 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
Enterprise Applications > IBM_Application_Center_Services > Security role to user/group mapping
Map Users...
Map Groups...
Map Special Subjects

Security role to user/group mapping

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

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

Figure 6-20. Mapping the **appcenteruser** *and* **appcenteradmin** *roles: user groups*

f. Customize as necessary.

g. Click **OK**.

h. Repeat steps 3c on page 6-245 to 3g to map the roles for the console web application; in step 3d on page 6-245, select **IBM_Application_Center_Console**.

i. Click **Save**.

4. If you deployed an EAR file, map the **`appcenteruser`** and **`appcenteradmin`** roles to a set of users as follows:

   a. Select **Applications** > **Application Types** > **WebSphere application servers**.

   b. Click **AppCenterEAR**.

   c. In the Detail Properties section, click **Security role to user/group mapping**.

   d. Customize as necessary.

   e. Click **OK**.

   f. Click **Save**.

Configuring the Java EE security roles on WebSphere Application Server Liberty profile:

Configure the Java EE security roles of the Application Center and the data source in the `server.xml` file.

Before you begin

Review the definition of roles at “Configuring user authentication for Application Center” on page 6-243.

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, 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>
   ```

   You must include this example in the following location: :

   - If you deployed WAR files, in the <application-bnd> element of each <application> element: the appcenterconsole and applicationcenter applications.
   - If you deployed an EAR file, in the <application-bnd> element of the applicationcenter application.

   Replace the <security-role> elements that have been created during installation for test purposes.

   ```
   <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>
   ```

   This example shows a definition of users and groups in the basicRegistry of WebSphere Application Server Liberty. For more information about configuring a user registry for WebSphere Application Server Liberty profile, see Configuring a user registry for the Liberty profile.
2. Edit the server.xml file to define the **AppCenterPool** size.

   ```
   <connectionManager id="AppCenterPool" minPoolSize="10" maxPoolSize="40"/>
   ```
3. In the <dataSource> element, define a reference to the connection manager:

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

Configuring the Java EE security roles on Apache Tomcat:

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

Before you begin

Review the definition of roles at “Configuring user authentication for Application Center” on page 6-243.

Procedure

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

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

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

Managing users with LDAP

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

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

If you plan to use an LDAP registry with the Application Center, you must configure your WebSphere Application Server or your Apache Tomcat server to use an LDAP registry to authenticate users.

In addition to authentication of users, configuring the Application Center for LDAP also enables you to use LDAP to define the users and groups who can install mobile applications through the Application Center. The means of defining these users and groups is the Access Control List (ACL).

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

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

LDAP with WebSphere Application Server V7:

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

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

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

Configuring LDAP authentication for WebSphere Application Server V7:

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

About this task

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

Procedure

1. Log in to the WebSphere Application Server console.
2. In **Security** > **Global Security**, verify that administrative security and application security are enabled.
3. Select **Federated repositories** or **Standalone LDAP registry**.
4. Click **Configure**. For federated repositories, follow step 5. For stand-alone LDAP registry, follow step 6
5. **Option for federated repositories**: add the new repository and configure the necessary additional properties.
   a. To add a 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 that are 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. If you deployed WAR files, 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**.

      You can then 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 access to the Application Center to everyone in the WebSphere user repository, including everyone registered in the LDAP registry
9. Repeat step 8 for **`IBM_Application_Center_Console`**.

   Make sure that in step 8.b you select **IBM_Application_Center_Console** instead of **IBM_Application_Center_Services**.
10. If you deployed an EAR file, in the WebSphere Application Server console, map the security roles to users and groups.
    a. Click **Applications** > **Application Types** > **WebSphere enterprise applications**.
    b. From the list of applications, click **AppCenterEAR**.

   c. In the Detail Properties section, click **Security role to user/group mapping**.
   d. For appcenteradmin and appcenteruser roles, select **Map groups** or **Map users** to select users or 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 access to the Application Center to everyone in the WebSphere user repository, including everyone registered in the LDAP registry.
11. Click **Save** to save your changes.

Configuring LDAP ACL management with JNDI for 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.

Note: 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** if you deployed WAR files or **APPCenterEAR** if you deployed an EAR file.
4. In the **Web Module Properties** section, select **Environment entries for Web modules**.
   a. For the **ibm.appcenter.ldap.vmm.active** entry, assign the falsevalue.
   b. For the **ibm.appcenter.ldap.active** entry, assign the true value.
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 become visible only 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 who is permitted to search the LDAP directory.
   - **ibm.appcenter.ldap.security.bindpwd**: the password of the user who is 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 does not handle LDAP referrals. Possible values:
   - **ignore**: ignores the 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 that is found in the LDAP server.

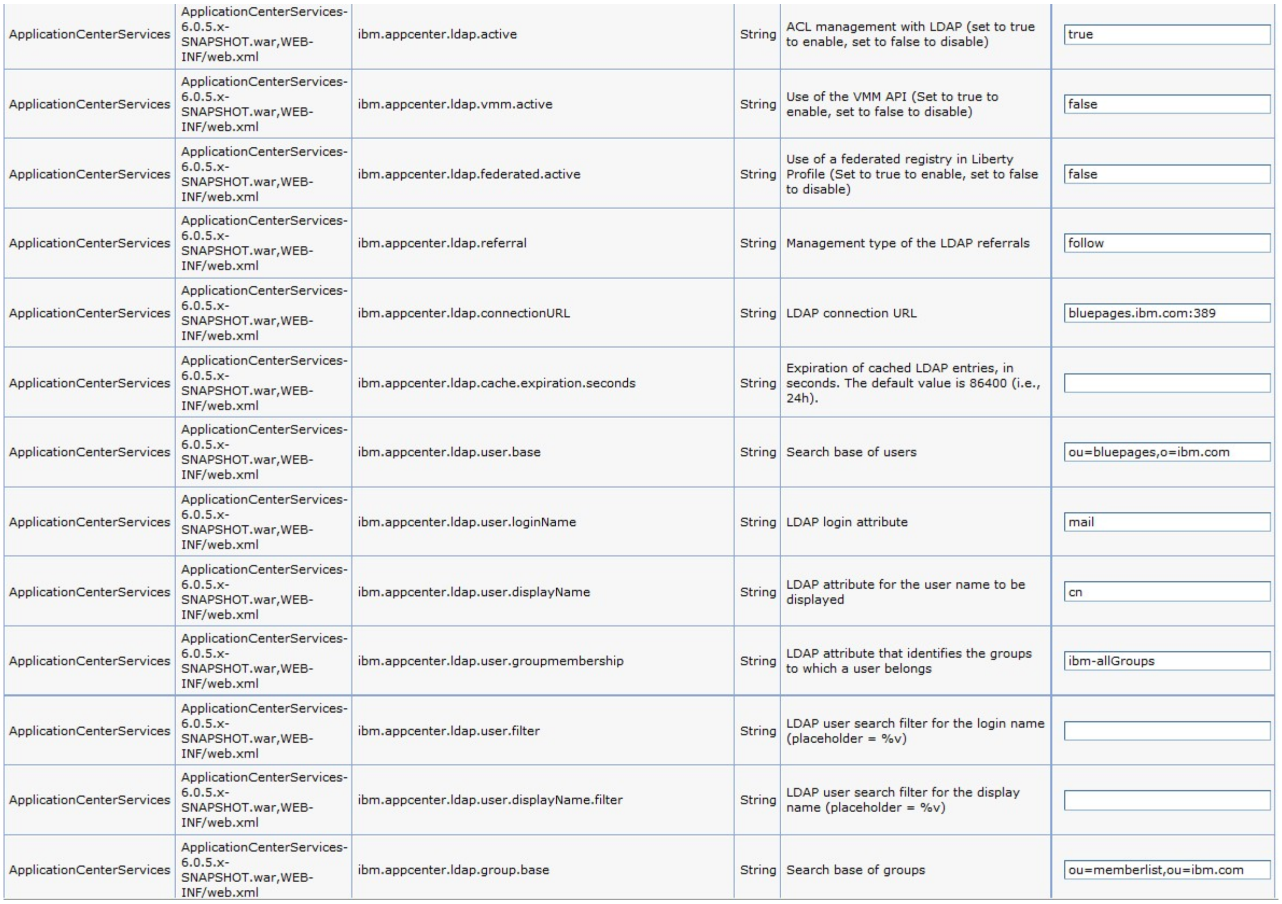
   a. Enter the value of the property and click **OK**.
   b. Save the configuration.
9. **Option:** *If users and groups are defined in the same subtree (the properties **ibm.appcenter.ldap.user.base** and **ibm.appcenter.ldap.group.base** have the same value), follow this step.* Configure the following 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 6-280 for a complete list of properties.



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 6-21. Environment entries and their values (LDAP and WebSphere Application Server V7)

Configuring LDAP ACL management with VMM for 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.

Note: 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.

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

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

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

   Where *your LDAP id* is the user ID that is 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 supported only 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 6-280 for a complete list of properties.

     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 become visible only 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
     ```

For more information, see Using the stand-alone tool to clear the LDAP cache.

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** if you deployed WAR files or **APPCenterEAR** if you deployed an EAR file, 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 pwd.txt file 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 for 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 repository and configure it.
   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 that are 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. If you deployed WAR files, 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 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. If you deployed an EAR file, in the WebSphere Application Server console, map the security roles to users and groups.
   a. Click **Applications** > **Application Types** > **WebSphere enterprise applications**.

   b. From the list of applications, click **AppCenterEAR**.
   c. In the Detail Properties section, click **Security role to user/group mapping**.
   d. For `appcenteradmin` and `appcenteruser` roles, select **Map groups** or **Map users** to select users or 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 access to the Application Center to everyone in the WebSphere user repository, including everyone registered in the LDAP registry.

10. Click **Save** to save your changes.

What to do next

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

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

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

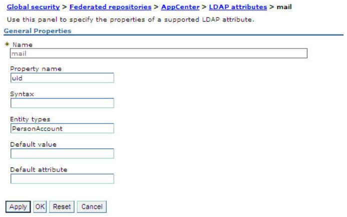
About this task

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

Then you should enable ACL management with VMM. You can configure LDAP based on the federated repository configuration only.

Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Security** > **Global security**.
3. In the "User account repository" section, select **Configure**.
4. Select your LDAP repository entry.
5. Under **Additional Properties**, select **LDAP attributes** (WebSphere Application Server V8.0) or **Federated repositories property names to LDAP attributes mapping** (WebSphere Application Server V8.5).
6. Select **Add** > **Supported**.
7. Enter these property values:
   a. For **`Name`** enter your LDAP login attribute.
   b. For **`Property name`** enter **`uid`**.
   c. For **`Entity types`** enter the LDAP entity type.
   d. Click **OK**.



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

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

General Properties

* Name: mail

Property name: uid

Syntax:

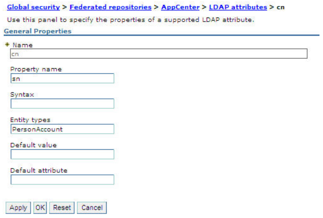
Entity types: PersonAccount

Default value:

Default attribute:

Figure 6-22. Associating LDAP login with uid property (WebSphere Application Server V8.0)

8. Select **Add** > **Supported**.
   a. For **`Name`** enter your LDAP attribute for full user name.
   b. For **`Property name`** enter **`sn`**.
   c. For **`Entity types`** enter the LDAP entity type.
   d. Click **OK**.



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

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

General Properties

* Name: cn

Property name: sn

Syntax:

Entity types: PersonAccount

Default value:

Default attribute:

Figure 6-23. Associating LDAP full user name with sn property (WebSphere Application Server V8.0)

9. Select **Add** > **Supported** to configure a group name:
   a. For **`Name`** enter the LDAP attribute for your group name.
   b. For **`Property name`** enter **`cn`**.
   c. For **`Entity types`** enter the LDAP entity type.
   d. Click **OK**.
10. Enable ACL management with LDAP:
   a. Select **Servers** > **Server Types** > **WebSphere application servers**.
   b. Select the appropriate application server.

      In a clustered environment you must configure all the servers in the cluster in the same way.
   c. In the **Configuration** tab, under “Server Infrastructure”, click the **Java and Process Management** tab and select **Process definition**.
   d. In the **Configuration** tab, under “Additional Properties”, select **Java Virtual Machine**,
   e. In the **Configuration** tab, under “Additional Properties”, select **Custom properties**.
   f. Enter the required property-value pairs in the form. To enter each pair, click **New**, enter the property and its value, and click **OK**.

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

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

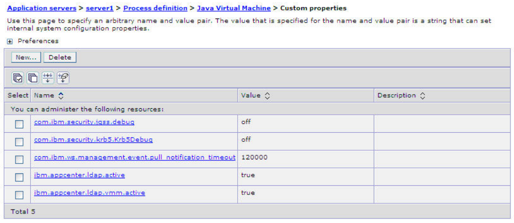
Changes to users and groups on the LDAP server become visible to the Application Center after a delay, which is specified by **ibm.appcenter.ldap.cache.expiration.seconds**. The Application Center maintains a cache of LDAP data and the changes only become visible after the cache expires. By default,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

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

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

Results

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



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

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

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

Figure 6-24. 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 for the 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 that are 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.
   - If you deployed WAR files: **`applicationcenter`** and **`appcenterconsole`**
   -

     If you deployed an EAR file:**`applicationcenter`**

   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. The **access-id** attribute must refer to the realm name that is 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.

```
<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>
```

If applicable, 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, make sure that the following feature is 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:

The `JNDI_property_name` argument is the name of the property that you are adding.

The `property_value` argument is the value of the property that you are adding.

Table 6-53. JNDI properties for configuring ACL management with LDAP in the server.xml file

Property	Description
ibm.appcenter.ldap.active	Set it to `true` to enable LDAP or to `false` to disable LDAP.
ibm.appcenter.ldap.federated.active	Since WebSphere Application Server Liberty profile V8.5.5: set it to `true` to enable use of the federated registry; set to `false` to disable use of the federated registry, which is the default setting.

Table 6-53. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
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 required only 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 required only 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 required only 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	This property identifies the distinguished name of the user who is allowed to search the LDAP directory. Use this property only if security binding is required.

Table 6-53. JNDI properties for configuring ACL management with LDAP in the server.xml file (continued)

Property	Description
`ibm.appcenter.ldap.security.bindpwd`	This property identifies the password of the user who 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 that is generated by the tool. The supported encoding types are `xor` and `aes` (only with the default key). 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 become visible only 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` See Using the stand-alone tool to clear the LDAP cache for details.
`ibm.appcenter.ldap.referral`	This property indicates whether referrals are supported by the JNDI API. If no value is passed, the JNDI API handles no LDAP referrals. Valid values: • `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 6-280 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=inetOrgPer
<jndiEntry jndiName="ibm.appcenter.ldap.user.displayName.filter" value='"(&amp;(cn=%v)(objectclass
<jndiEntry jndiName="ibm.appcenter.ldap.group.filter" value='"(&amp;(cn=%v)(|(objectclass=groupOfN
```

LDAP with Apache Tomcat:

Configure the Apache Tomcat server for LDAP authentication and configure security (Java™ Platform, Enterprise Edition) in the `web.xml` file of the Application Center.

To configure ACL management of the Application Center; configure LDAP for user authentication, map the Java Platform, Enterprise Edition (Java EE) 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.

Configuration of 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, follow this process:

1. Configure LDAP for user authentication.
2. Map the Java Platform, Enterprise Edition (Java EE) roles of the Application Center to the LDAP roles.
3. Configure theApplication Center properties for LDAP authentication.

Restriction: 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 theApplication 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. For more information about configuring a realm, see the Realm Component on the Apache Tomcat website.

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 that is 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.

Set the value of **userSubtree** to `true`; if it is set to `false`, the search runs 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 Java EE security roles. Therefore, you must map LDAP attributes to some Java EE 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 that is shared by all authenticated users. This attribute is optional.

Mapping the Java EE roles of the Application Center to LDAP roles

After you define the LDAP request for the Java EE 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 Java EE 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. Replace the names MyLdapAdmin and MyLdapUser with the roles that are defined in your LDAP. Modify the following files:

- *tomcat_install_dir*/webapps/appcenterconsole/WEB-INF/web.xml
- *tomcat_install_dir*/webapps/applicationcenter/WEB-INF/web.xml

The security-role-ref element in the JAX_RS servlet

```
<servlet>
    <servlet-name>...</servlet-name>
    <servlet-class>...</servlet-class>
        <init-param>
            ...
        </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>
    <security-role>
        <role-name>MyLdapUser</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 that are defined in the auth-constraint elements to protect the web resources. Edit these roles for the appcenteradminConstraint element in both the web.xml file of both appcenterconsole and applicationcenter, and for the appcenteruserConstraint element in the appcenterconsole web.xml file.

```
<security-constraint>
        <display-name>appcenteradminConstraint</display-name>
        <web-resource-collection>
            ...
        </web-resource-collection>
        <auth-constraint>
            <role-name>MyLdapAdmin</role-name>
        </auth-constraint>
        <user-data-constraint>
            ...
        </user-data-constraint>
</security-constraint>
```

and

```
<security-constraint>
        <display-name>appcenteruserConstraint</display-name>
        <web-resource-collection>
            ...
        </web-resource-collection>
        <auth-constraint>
            <role-name>MyLdapUser</role-name>
        </auth-constraint>
        <user-data-constraint>
            ...
        </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 must have the following syntax:

```
<Environment name="JNDI_property_name" value="property_value" type="java.lang.String" override="fals
```

Where:

- The `JNDI_property_name` parameter is the name of the property that you are adding.
- The `property_value` parameter sis the value of the property that you are adding.

Table 6-54. 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`.

Table 6-54. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat (continued)

Property	Description
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".
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.

Table 6-54. Properties for configuring ACL management for LDAP in the server.xml file on Apache Tomcat (continued)

Property	Description
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 -` 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 6-280 for a complete list of LAPD properties that you can set.

Example

The example shows LDAP properties that are 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.la
<Environment name="ibm.appcenter.ldap.user.base" value="dc=ibm,dc=com" type="java.lang.String" overr
<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" override=
<Environment name="ibm.appcenter.ldap.user.groupmembership" value="ibm-allGroups" type="java.lang.St
<Environment name="ibm.appcenter.ldap.group.base" value="dc=ibm,dc=com" type="java.lang.String" over
<Environment name="ibm.appcenter.ldap.group.name" value="cn" type="java.lang.String" override="false
<Environment name="ibm.appcenter.ldap.group.uniquemember" value="uniquemember" type="java.lang.Strin
<Environment name="ibm.appcenter.ldap.cache.expiration.seconds" value="43200" type="java.lang.String
<Environment name="ibm.appcenter.ldap.security.sasl" value="EXTERNAL" type="java.lang.String" overri
<Environment name="ibm.appcenter.ldap.security.referral" value="follow" type="java.lang.String" over
<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)
```

Configuring properties of DB2 JDBC driver in WebSphere Application Server

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

About this task

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

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

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

Procedure

1. 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.

Managing the DB2 transaction log size

When you upload an application that is at least 40 MB with IBM MobileFirst Platform 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 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 MobileFirst 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.

Defining the endpoint of the application resources

When you add a mobile application from the Application Center console, the server-side component creates Uniform Resource Identifiers (URI) for the application resources (package and icons). The mobile client uses these URI to manage the applications on your device.

Purpose

To manage the applications on your device, the Application Center console must be able to locate the Application Center REST services and to generate the required number of URI that enable the mobile client to find the Application Center REST services.

By default, the URI protocol, host name, and port are the same as those defined in the web application server used to access the Application Center console; the context root of the Application Center REST services is `applicationcenter`. When the context root of the Application Center REST services is changed or when the internal URI of the web application server is different from the external URI that can be used by the mobile client, the externally accessible endpoint (protocol, host name, and port) of the application resources must be defined by configuring the web application server. (Reasons for separating internal and external URI could be, for example, a firewall or a secured reverse proxy that uses HTTP redirection.)

The following figure shows a configuration with a secured reverse proxy that hides the internal address (192.168...). The mobile client must use the external address (**`appcntr.net`**).

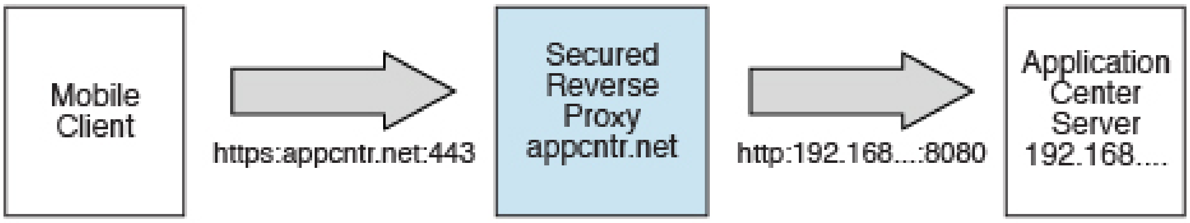

Mobile Client
https:appcntr.net:443
Secured Reverse Proxy appcntr.net
http:192.168...:8080
Application Center Server 192.168....

Figure 6-25. Configuration with secured reverse proxy

Table 6-55. 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 6-280 for a complete list of endpoint properties that you can set.

Configuring the endpoint of 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. The procedure differs depending on whether you deployed WAR files or an EAR file.

If you deployed WAR files:

About this task

Follow this procedure when you must change the URI protocol, host name, and port used by the mobile client to manage the applications on your device. Since IBM Worklight V6.0, you use JNDI environment entries.

For a complete list of JNDI properties, see "List of JNDI properties for the Application Center" on page 6-280.

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 host name.
   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 the context root.
10. Click **OK** and save the configuration.

If you deployed an EAR file:
Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Applications** > **Application Types** > **WebSphere enterprise applications**.
3. Click **AppCenterEAR**.
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 host name.
   b. For **`ibm.appcenter.proxy.port`**, assign the port number.
   c. For **`ibm.appcenter.proxy.protocol`**, assign the external protocol.
6. 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 the context root.

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

Configuring the endpoint of the application resources (Liberty profile):

For the Liberty profile, configure the endpoint of the application resources through the JNDI environment.

Purpose

Since IBM Worklight V6.0, follow this procedure when you must change the URI protocol, host name, 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 that you are adding.

`property_value` is the value of the property that you are adding.

Table 6-56. 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. 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 host name 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 6-280.

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/applicationcen
<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, host name, 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 6-57. 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 host name of the application resources URI.

Table 6-57. Properties in the server.xml file for configuring the endpoint of the application resources (continued)

Property	Type	Description
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 6-280.

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"
type="java.lang.String" override="false"/>
```

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/applicationcenter"
type="java.lang.String" override="false"/>
<Environment name="ibm.appcenter.proxy.protocol" value="https" type="java.lang.String"
override="false"/>
<Environment name="ibm.appcenter.proxy.host" value="appcntr.net" type="java.lang.String"
override="false"/>
<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.

For iOS applications, you must configure the Application Center server with SSL and enable the TLS v1.2 protocol.

SSL transmits data over the network in a secured channel. You must purchase an official SSL certificate from an SSL certificate authority. 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 iOS 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 the iOS operating systems do not support this confirmation dialog in system calls. This limitation is a reason to avoid self-signed certificates or SSL certificates that are not suited to mobile operating systems. On the iOS operating system, 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 6-180 and "Managing and installing self-signed CA certificates in an Application Center test environment" on page 6-278.

The following topics describe how to enable SSL on the application servers. If your server is behind a reverse proxy that decrypts the SSL-encoded packets before passing them the application server, you must configure this reverse proxy. For IBM HTTP Server, see the IHS documentation.

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**.

To enable TLS v1.2, which is required for iOS9, the collection of all SSL configurations is listed.

4. For each SSL configuration in the list, modify the SSL protocol to use SSL_TLSv2.
   a. Select an SSL Configuration and then, under **Additional Properties**, click **Quality of protection (QoP) settings**.
   b. In **Quality of protection (QoP) settings**, from **Protocol**, select **SSL_TLSv2**.
   c. Select **Apply** and then **Save**.
   d. Make sure that the JRE supports the TLS v1.2 protocol. Use a JRE described in IBM Security Bulletin Vulnerability in SSLv3 affects IBM WebSphere Application Server (CVE-2014-3566) or later.

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. Enable TLS v1.2, which is required for iOS9.
   a. If you use an IBM Java SDK, add the following line to the `server.xml` file:

      ```
      <ssl id="defaultSSLConfig" keyStoreRef="defaultKeyStore" sslProtocol="SSL_TLSv2"/>
      ```
   b. To all existing `<ssl>` elements, add `sslProtocol="SSL_TLSv2"`. You must use a JRE that supports the TLS v1.2 protocol. Use a JRE described in IBM Security Bulletin Vulnerability in SSLv3 affects IBM WebSphere Application Server (CVE-2014-3566) or later, or Oracle JRE 1.7.0_75 or later, or Oracle JRE 1.8.0_31 or later.
7. 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. To enable TLS v1.2, which is required for iOS9, add the following attribute to the `<Connector>` element:
   ```
   sslEnabledProtocols="TLSv1.2,TLSv1.1,TLSv1,SSLv2Hello"
   ```

   The result should be similar to this example:
   ```
   <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
              maxThreads="150" scheme="https" secure="true"
              clientAuth="false" sslProtocol="TLS"
              keystoreFile="/path/to/keystore.jks"
              sslEnabledProtocols="TLSv1.2,TLSv1.1,TLSv1,SSLv2Hello"
              keystorePass="mypassword" />
   ```
   You must use a JRE that supports TLS v1.2.
   - Oracle JRE 1.7.0_75 or later
   - Oracle JRE 1.8.0_31 or later
5. Restart the web server. Now you can access the web server by `https://`*`myserver`*`:8443/...`

Managing and installing self-signed CA certificates in an Application Center test environment:

Use self-signed certificate authority (CA) certificates in test environments to install applications with Application Center on a mobile device from a secured server.

Uploading or deleting a certificate:

Before you begin

When you install the Application Center mobile client from OTA (the bootstrap page), the device user must upload and install the self-signed CA file before the Application Center mobile client is installed.

About this task

When you use Application Center for a test installation, the administrator might not have a real Secure Sockets Layer (SSL) certificate available. You might want to use a self-signed CA certificate. Such certificates work if they get installed on the device as root certificate. For the basic concepts of CA certificates and further details about such certificates, see “Configuring SSL by using untrusted certificates” on page 6-180.

As an administrator, you can easily distribute self-signed CA certificates to devices.

Support for X.509 certificates comes from the iOS platform, not from IBM MobileFirst Platform Foundation for iOS. For more information about specific requirements for X.509 certificates, see iOS documentation.

Procedure

Managing self-signed certificates: in your role of administrator of Application Center, you can access the list of registered self-signed CA certificates to upload or delete certificates.

1. To display Application Center settings, click the gear icon ⚙ .
2. To display the list of registered certificates, select **Self Signed Certificates**.
3. Upload or delete a certificate.
   - To upload a self-signed CA certificate, in the Application Center console, click **Upload a certificate** and select a certificate file.

     Note: The certificate file must be in PEM file format. Typical file name suffixes for this type of file are `.pem`, `.key`, `.cer`, `.cert`. The certificate must be a self-signed one, that is, the values of the **Issuer** and **Subject** fields must be the same. And the certificate must be a CA certificate, that is, it must have the X509 extension named `BasicConstraint` set to `CA:TRUE`.
   - To delete a certificate, click the trash can icon on the right of the certificate file name in the list.

Installing a self-signed CA certificate on a device:

About this task

Registered self-signed CA certificates are available through the bootstrap page at `http://`*`hostname:portnumber`*`/appcenterconsole/installers.html`

Where:

- *hostname* is the name of the server that hosts the Application Center console.
- *portnumber* is the corresponding port number.

Procedure

1. Click the **SSL Certificates** tab.
2. To display the details of a certificate, select the appropriate registered certificate.
3. To download and install the certificate on the device, click **Install**.

Troubleshooting SSL:

Learn how to troubleshoot SSL configurations.

The topics in this section describe how to resolve various situations that might occur in different SSL configurations.

Incompatibility between SSL certificate and mobile devices:

Resolve the situation when an SSL certificate is not compatible with mobile devices.

Symptoms
You cannot connect to the web server, either through the browser when you open the `installers.html` page, or through the mobile client; you cannot download any application from the Application Center web server.

Causes
Different mobile devices recognize different trusted root certificates.

Resolving the problem
User response: Contact the certificate authority for information about compatibility of the certificate with mobile devices.

SSL certificate is incompatible with browser, but compatible with mobile operating system:

Resolve the situation where you cannot install applications from the Application Center website, but can see the website.

Symptoms
You cannot connect to the web server, either through the browser when you open the `installers.html` page, or through the mobile client; you cannot download any application from the Application Center web server.

Causes
The installation procedure is done by the operating system; the trusted root certificates of the operating system might not be the same as the trusted root certificates of the browser.

Resolving the problem
User response: Contact the certificate authority for information about compatibility of the certificate with mobile devices.

List of JNDI properties for the Application Center

You can configure JNDI properties for the Application Center.

Table 6-58. 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`**.

Table 6-58. List of the JNDI properties for the Application Center (continued).

Property	Description
appcenter.openjpa.ConnectionPassword	The password for the database connection. This property is only needed when the database is not specified in **appcenter.jndi.name**.
appcenter.openjpa.ConnectionURL	The URL specific to the database connection driver class. This property is only needed when the database is not specified in **appcenter.jndi.name**.
appcenter.openjpa.ConnectionUserName	The user name or the database connection. This property is only needed when the database is not specified in **appcenter.jndi.name**.
ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate	Specifies whether the certificate that enables Application Center to send push notifications about updates of iOS applications is a development certificate. Set to `true` to enable or `false` to disable. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 12-74.
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 12-74.
ibm.appcenter.apns.p12.certificate.password	The password of the certificate that enables Application Center to send push notifications about updates of iOS applications. See "Configuring the Application Center server for connection to Apple Push Notification Services" on page 12-74.
ibm.appcenter.forceUpgradeDBTo60	The database design was changed starting from IBM 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.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 12-105.
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 6-248.

Table 6-58. 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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-248.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.

Table 6-58. 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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 (only with the default key). 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 6-266.
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 6-266.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.

Table 6-58. 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 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
ibm.appcenter.ldap.user.loginName	The login name attribute when you use LDAP without VMM. See "Configuring LDAP ACL management (Liberty profile)" on page 6-260 and "Configuring LDAP ACL management (Apache Tomcat)" on page 6-266.
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 6-256 and "Configuring LDAP ACL management with VMM for WebSphere Application Server V7" on page 6-252.
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 6-256 and "Configuring LDAP ACL management with VMM for WebSphere Application Server V7" on page 6-252.
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 6-256 and "Configuring LDAP ACL management with VMM for WebSphere Application Server V7" on page 6-252.
ibm.appcenter.logging.formatjson	This property has only an effect when **ibm.appcenter.logging.tosystemerror** is set to true. If enabled, it formats JSON responses in logging messages that are directed to System.Error. Setting this property is helpful when you debug the server.

Table 6-58. List of the JNDI properties for the Application Center (continued).

Property	Description
`ibm.appcenter.logging.tosystemerror`	Specifies whether all logging messages are also directed to System.Error. Setting this property is helpful when you debug the server.
`ibm.appcenter.openjpa.Log`	This property is passed to OpenJPA and enables JPA logging. For details, see the Apache OpenJPA User's Guide.
`ibm.appcenter.proxy.host`	If the Application Center server is behind a firewall or reverse proxy, this property specifies the address of the host. Setting this property allows a user outside the firewall to reach the Application Center server. Typically, this property is the address of the proxy. See "Defining the endpoint of the application resources" on page 6-270.
`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 6-270.
`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 6-270.
`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 6-270.

Table 6-58. List of the JNDI properties for the Application Center (continued).

Property	Description
`ibm.appcenter.services.iconCacheMaxAge`	Specifies the amount of time in seconds cached icons remain valid for the Application Center console and the Client. Application icons rarely change, therefore they are cached. Specify values larger than 600 (10 min) to reduce the amount of data transfer for the icons.
`ibm.worklight.jndi.configuration`	Optional. If the JNDI configuration is injected into the WAR files or provided as a shared library, the value of this property is the name of the JNDI configuration. This value can also be specified as a system property. See "Predefining MobileFirst Server configuration for several deployment environments" on page 6-287.
`ibm.worklight.jndi.file`	Optional. If the JNDI configuration is stored as an external file, the value of this property is the path of a file that describes the JNDI configuration. This value can also be specified as a system property. See "Predefining MobileFirst Server configuration for several deployment environments" on page 6-287.

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 Apple iTunes must be imported into the WebSphere truststore before you can use application links to iTunes.

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

Configuring WebSphere Application Server to support applications in 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 truststore. 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)
```

Predefining MobileFirst Server configuration for several deployment environments

You can configure JNDI properties in a property file for easy transfer between one web application server and another; for example, to transfer from test to production environments.

As part of the installation of administration components of IBM MobileFirst Platform Foundation for iOS, various JNDI properties must be set. These components include MobileFirst Operations Console, MobileFirst Administration

Service, and Application Center. Normally, JNDI properties are specified in the configuration of the web application server and are outside the web archive (WAR) file that represents the server component.

Instead, you can specify the JNDI properties in a property file. Having JNDI properties in a property file makes it easier to transfer the entire configuration from one web application server to another. For example, you can configure a test web server and, once the configuration is stable, you can transfer the configuration to the production web server by copying the property file to the production server.

This property file can be made available to the server components in various ways:

- The property file can be placed on the file system.

  This solution is particularly useful for a stand-alone test server when you are experimenting with JNDI properties to determine the final configuration. You can easily change the file on the file system with a text editor. Then you have only to restart the web server to enable the changed configuration.
- The property file can be injected into web archive (WAR) files.

  This solution is useful when you want to transfer the configuration together with the web archive file to another web server. You only have to handle the web archive file, and no other files. The configuration is, in this case, fused into the web archive file.
- The property file can be installed as a shared library for all server components.

  This solution is useful when you intend to exchange the web archive files often, but want to keep the same configuration all the time.

Creating the property file

Define JNDI properties in a property file by using a text editor. Determine where to set JNDI properties according to a selective priority.

The property file follows the standard Java property file syntax and can be edited with any text editor. It has the file extension `.properties`. You can include all properties of all web archive (WAR) files in the same property file.

Here is an example of the content of a property file.

```
publicWorkLightHostname=myworklighthost.net
publicWorkLightPort=9080
publicWorkLightProtocol=https
push.gcm.proxy.enabled=false
push.gcm.proxy.host=myproxyhost.net
push.gcm.proxy.port=-1
push.gcm.proxy.protocol=https
ibm.worklight.admin.environmentid=id123
```

JNDI properties

You can refer to the details of JNDI properties in the relevant parts of the user documentation:

Application Center
"List of JNDI properties for the Application Center" on page 6-280

MobileFirst Application Services
"List of JNDI properties for MobileFirst Server administration" on page 6-101

MobileFirst runtime
“Configuration of MobileFirst applications on the server” on page 11-50

You do not have to specify all the possible JNDI properties in the property file. You can specify some in the property file and others as JNDI properties that are explicitly set in the web application server. The following list indicates the priority by which properties are enacted.

1. If a JNDI property is explicitly set in the web application server, this property value is taken. Refer to the documentation of your web application server for how to set JNDI properties.
2. If that is not the case, but the JNDI property is set in the property file injected into the web archive file or in the property file provided as a shared library, the property value is taken from this property file.
3. If that is not the case, but the JNDI property is set in the property file provided on the file system, the property value is taken from this property file.
4. If that is not the case, the default value of the JNDI property is taken.

Using a property file in the file system

You can place the property file directly into the file system of the web application server.

The property file can be stored directly in the file system. This approach is particularly useful for a stand-alone test server when you are experimenting with JNDI properties to determine the final configuration. You can easily change the file on the file system with a text editor. Then you have only to restart the web server to enable the changed configuration.

You must define the property **`ibm.worklight.jndi.file`** to point to the location of the property file. This property can be defined as a Java Virtual Machine system property or explicitly as a JNDI property. This property cannot be defined in the property file.

WebSphere Application Server full profile

Determine a suitable directory for the JNDI property file in the WebSphere Application Server installation directory.

- For a stand-alone server, you can use a directory such as:
  $WAS_INSTALL_DIR/profiles/*profile-name*/config/*mywlconfig*.properties
- For deployment to a WebSphere Application Server ND cell, use for example:
  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/*mywlconfig*.properties
- For deployment to a WebSphere Application Server ND cluster, use for example:
  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/*mywlconfig*.properties
- For deployment to a WebSphere Application Server ND node, use for example:
  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/*mywlconfig*.properties
- For deployment to a WebSphere Application Server ND server, use for example:
  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/servers/*server-name*/*mywlconfig*.properties

Next, add the setting of the **ibm.worklight.jndi.file** property to the Java Virtual Machine custom properties in the WebSphere Application Server administration console. For details of how to add this setting, see "Setting the file pointer property (WebSphere Application Server full profile)."

WebSphere Application Server Liberty profile

You must place the property file inside the server directory of the Liberty server; for example, place it in $LIBERTY_HOME/usr/servers/worklightServer/*mywlconfig*.properties.

Edit the $LIBERTY_HOME/usr/servers/worklightServer/bootstrap.properties file and add the property **ibm.worklight.jndi.file** to point to the property file; for example, ibm.worklight.jndi.file=mywlconfig.properties.

Alternatively, instead of editing bootstrap.properties, create or edit the file $LIBERTY_HOME/usr/servers/worklightServer/jvm.options and add, for example:

```
-Dibm.worklight.jndi.file=mywlconfig.properties
```

You can use the normal web archive (WAR) files of the web applications. They can be installed as described elsewhere in "Installing and configuring" on page 6-1.

Restart the web application server. Whenever the property file changes, the web application server must be restarted.

Apache Tomcat

You must place the property file inside the conf directory of the Apache Tomcat server; for example, place it in $TOMCAT_HOME/conf/*mywlconfig*.properties.

Edit the $TOMCAT_HOME/conf/catalina.properties file and add the property **ibm.worklight.jndi.file** to point to the property file; for example, **ibm.worklight.jndi.file**=../conf/mywlconfig.properties.

Alternatively, on UNIX systems, instead of editing catalina.properties, create or edit the $TOMCAT_HOME/bin/setenv.sh file and add, for example:

```
CATALINA_OPTS="$CATALINA_OPTS
-Dibm.worklight.jndi.file=../conf/mywlconfig.properties"
```

or on microsoft Windows systems, create or edit the $TOMCAT_HOME/bin/setenv.bat file and add , for example:

```
set CATALINA_OPTS=%CATALINA_OPTS%
-Dibm.worklight.jndi.file=../conf/mywlconfig.properties
```

You can use the normal web archive (WAR) files of the web applications. They can be installed as described elsewhere in "Installing and configuring" on page 6-1.

Restart the web application server. Whenever the property file changes, the web application server must be restarted.

Setting the file pointer property (WebSphere Application Server full profile)

Define the **ibm.worklight.jndi.file** property through the administration console of the WebSphere Application Server full profile.

Before you begin

Determine the location in the file system of the JNDI property file. See "WebSphere Application Server full profile" on page 6-289.

About this task

When you opt to configure JNDI properties by using a property file located directly in the file system, you must set a property to point to the property file. This property is outside the property file and is set through the administration console.

You must log in to the WebSphere Application Server administration console and add the setting of the **`ibm.worklight.jndi.file`** property to the Java Virtual Machine custom properties.

Procedure

1. Select **Servers** > **Server types** > **WebSphere Application Servers**.
2. Select the name of your server.
3. Expand "Java and process management" and select "Process Definition".
4. Select "Java Virtual Machine".
5. Select "Custom Properties" and click **New**.
6. Specify the name as **`ibm.worklight.jndi.file`**.
7. Specify the value as the path to the property file. The directory `$WAS_INSTALL_DIR/profiles/profile-name` can be specified as `${USER_INSTALL_ROOT}`; for example, that can be one of the following values:
   - `${USER_INSTALL_ROOT}/config/`*`mywlconfig`*`.properties`
   - `${USER_INSTALL_ROOT}/config/cells/`*`cell-name`*`/`*`mywlconfig`*`.properties`
   - `${USER_INSTALL_ROOT}/config/cells/`*`cell-name`*`/clusters/`*`cluster-name`*`/`*`mywlconfig`*`.properties`
   - `${USER_INSTALL_ROOT}/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/`*`mywlconfig`*`.properties`
   - `${USER_INSTALL_ROOT}/config/cells/`*`cell-name`*`/nodes/`*`node-name`*`/servers/`*`server-name`*`/`*`mywlconfig`*`.properties`
8. Click **Apply**.
9. Click **Save**.

What to do next

You can use the normal web archive (WAR) files of the web applications. They can be installed as described elsewhere in "Installing and configuring" on page 6-1.

To enable the property file, restart all MobileFirst Server components. These components are displayed in the WebSphere Application Server administration console under WebSphere enterprise applications.

Using property files injected into a web archive file

You can inject several configuration files into the WAR file of a MobileFirst Server component.

The property file can be injected into the web archive (WAR) files for MobileFirst Operations Console, MobileFirst Administration Service, MobileFirst runtime, or

Application Center. This approach is useful when you want to transfer the configuration together with the web archive file to another web application server. In this case, you only have to handle the web archive file, and no other files. The configuration is, in this case, fused into the web archive file.

You can inject several different configurations into the same web archive file and then select in the web application server which configuration should be used. For example, you could have a test configuration and a production configuration injected at the same time. To do so, create multiple property files with different settings, one named `testconf.properties` and the other named `prodconf.properties`.

Some JNDI properties must have the same value in all MobileFirst Server components. Therefore, you should inject the same property files into all web archive files. The following JNDI properties must be the same for MobileFirst Operations Console, MobileFirst Administration Services, and MobileFirst runtime:

- **`ibm.worklight.admin.environmentid`**
- **`ibm.worklight.topology.clustermode`**
- **`ibm.worklight.topology.platform`**
- **`ibm.worklight.admin.jmx.connector`**
- **`ibm.worklight.admin.jmx.dmgr.host`**
- **`ibm.worklight.admin.jmx.dmgr.port`**
- **`ibm.worklight.admin.jmx.host`**
- **`ibm.worklight.admin.jmx.port`**
- **`ibm.worklight.admin.jmx.user`**
- **`ibm.worklight.admin.jmx.pword`**
- **`ibm.worklight.admin.rmi.registryPort`**
- **`ibm.worklight.admin.rmi.serverPort`**

Injecting property files into a WAR file by using the Command Line tool

The wljndiinject command line tool is used to inject a set of property files into a web archive file. To add the property files `testconf.properties` and `prodconf.properties` to a war file, use the following command:

```
wljndiinject --sourceWarFile source.war testconf.properties prodconf.properties
```

The resulting web archive file can be found in the folder `jndi-injected`. It contains the property files inside the web archive file.

Options of the tool:

--help Shows the help.

--sourceWarFile *file*
The web archive file that is used to add the property files.

--destFile *file*
The destination file name. If not specified, the destination file is placed in the `jndi-injected` directory.

--sharedJar
Used to create a shared library; For details, see "Creating a shared library of JNDI properties" on page 6-295.

Injecting property files into a WAR file by using an Ant task

You can use the com.worklight.ant.jndi.JNDIInjectionTask Ant task to inject a set of property files into a web archive file.

Here is a sample ant script that shows the use of the ant task:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="WLJndiInjectTask" basedir="." default="jndiinject.Sample">
  <property name="install.dir" value="/path.to.worklight.installation" />
  <path id="classpath.run">
    <fileset dir="${install.dir}/WorklightServer/">
      <include name="worklight-ant-deployer.jar" />
    </fileset>
  </path>
  <target name="jndiinject.init">
    <taskdef name="jndiinject"
        classname="com.worklight.ant.jndi.JNDIInjectionTask">
      <classpath refid="classpath.run" />
    </taskdef>
  </target>
  <target name="jndiinject.Sample"
          description="Injects properties into the Worklight war file"
          depends="jndiinject.init">
    <!--  This is just an example:
        Mandatory parameters are sourceWarFile and the fileset.
        All other parameters are optional and could be ommitted.
        The source war files are expected in the wars directory.
        The property files are expected in the properties directory.
     -->
    <jndiinject
      sourceWarFile="wars/worklightproject.war"
      destWarFile="worklightproject-injected.war" >
        <fileset dir="." casesensitive="yes">
          <include name="properties/*.properties"/>
        </fileset>
    </jndiinject>
    <jndiinject
      sourceWarFile="wars/worklightadmin.war"
      destWarFile="worklightadmin-injected.war" >
        <fileset dir="." casesensitive="yes">
          <include name="properties/*.properties"/>
        </fileset>
    </jndiinject>
    <jndiinject
      sourceWarFile="wars/worklightconsole.war"
      destWarFile="worklightconsole-injected.war" >
        <fileset dir="." casesensitive="yes">
          <include name="properties/*.properties"/>
        </fileset>
    </jndiinject>
  </target>
</project>
```

Installing the property-injected WAR files in the web application server

After injection of the property files into the web archive files, the web archive files contain the property files and can be installed like any normal web archive file in the web application server.

For the MobileFirst Administration Services, MobileFirst Operations Console, and the MobileFirst runtime, you can use the ant task to install the web archive files, or you can update the web archive files manually.

For details of how to install web archive files for MobileFirst components, see:

- "Using Ant tasks to install MobileFirst Server administration" on page 6-62
- "Deploying a project WAR file and configuring the application server with Ant tasks" on page 11-16
- "Deploying the Application Center WAR files and configuring the application server manually" on page 6-234

When the web archive file is deployed, you must define the **ibm.worklight.jndi.configuration** property to point to the name of the required configuration.

Selecting the configuration in a property-injected WAR file

The default configuration is called `default.properties`. If the configuration of JNDI properties has a different name, you must define the **ibm.worklight.jndi.configuration** property. The value of this property must be the configuration name without the extension `.properties`. This property can be specified as a Java Virtual Machine system property or explicitly as a JNDI property. This property cannot be defined in the configuration property file.

Selecting the configuration: WebSphere Application Server full profile

You must log in to the WebSphere Application Server administration console and add the setting of the **ibm.worklight.jndi.configuration** property to the Java Virtual Machine custom properties.

To add this property setting:

1. Select **Servers** > **Server types** > **WebSphere Application Servers**.
2. Select the name of your server.
3. Expand "Java and process management" and select "Process Definition"
4. Select "Java Virtual Machine".
5. Select "Custom Properties" and click **New**.
6. Specify the name as **ibm.worklight.jndi.configuration**.
7. Specify the value as the name of the configuration.
8. Click **Apply**.
9. Click **Save**.

When the property is set, to enable the configuration, restart the appropriate MobileFirst Server components. These components are displayed in the WebSphere Application Server administration console under WebSphere enterprise applications.

Selecting the configuration: WebSphere Application Server Liberty profile

You must edit the *$LIBERTY_HOME*`/usr/servers/worklightServer/bootstrap.properties` file and set the **ibm.worklight.jndi.configuration** property to point to the name of the configuration.

For example:

```
ibm.worklight.jndi.configuration=testconf
```

Alternatively, instead of editing the `bootstrap.properties` file, create or edit the `$LIBERTY_HOME/usr/servers/worklightServer/jvm.options` file. For example, add:

```
-Dibm.ibm.worklight.jndi.configuration=testconf
```

To enable the configuration, restart the web application server.

Selecting the configuration: Apache Tomcat

You must edit the `*$TOMCAT_HOME*/conf/catalina.properties` file and set the **`ibm.worklight.jndi.configuration`** property to point to the name of the configuration.

For example:

```
ibm.worklight.jndi.configuration=testconf
```

Alternatively, instead of editing the `catalina.properties` file, depending on the operating system, create or edit one of the following files:

- On UNIX systems: `*$TOMCAT_HOME*/bin/setenv.sh`

  For example, add:

  ```
  CATALINA_OPTS="$CATALINA_OPTS -Dibm.worklight.jndi.configuration=testconf"
  ```

- On Microsoft Windows systems: *$TOMCAT_HOME*/bin/setenv.bat

  For example, add:

  ```
  set CATALINA_OPTS=%CATALINA_OPTS% -Dibm.worklight.jndi.configuration=testconf
  ```

To enable the configuration, restart the web application server.

Using a shared library of JNDI properties

You can create a shared library to hold different configurations for any MobileFirst Server component.

If you do not want to inject properties into web archive files, the property file can be installed as a shared library for all MobileFirst Server components. This approach is useful when you intend to exchange the web archive files often, but want to keep the same configuration all the time. The original web archive files remain unchanged, but you need to install an additional shared library.

You can add several different configurations to the same shared library and then select in the web application server which configuration to use. For example, you could have a test configuration and a production configuration injected at the same time. To do so, create property files with different settings, one named `testconf.properties` and the other `prodconf.properties`.

Creating a shared library of JNDI properties

The wljndiinject command line tool is used to create a shared library for a set of property files. To create a shared library named `jndiprops.jar` with the property files `testconf.properties` and `prodconf.properties`, use the following command:

```
wljndiinject --sharedJar --destFile jndiprops.jar testconf.properties prodconf.properties
```

Options of the tool:

--help Shows the help.

--sourceWarFile *file*

This option is not required for creating a shared library. This option is used when a property file is injected into a web archive file to identify the web archive file.

--destFile *file*

The destination file name of the shared library.

--sharedJar

Used to create a shared library instead of injecting a property file into a web archive file.

Installing a shared library of JNDI configurations

Assume that all web applications are already installed. The shared library is added to the web applications.

WebSphere Application Server full profile

Determine a suitable directory for the shared library `jndiprops.jar` in the WebSphere Application Server installation directory and place the `jndiprops.jar` file there.

- For a stand-alone server, you can use a directory such as:

  $WAS_INSTALL_DIR/optionalLibraries/IBM/Worklight
- For deployment to a WebSphere Application Server ND cell, use for example:

  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/Worklight
- For deployment to a WebSphere Application Server ND cluster, use:

  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/clusters/*cluster-name*/Worklight
- For deployment to a WebSphere Application Server ND node, use:

  $WAS_INSTALL_DIR/profiles/*profile-name*/config/cells/*cell-name*/nodes/*node-name*/Worklight
- 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

For details about adding the shared library, see "Adding the shared library (WebSphere Application Server full profile)" on page 6-298.

WebSphere Application Server Liberty profile

Place the `jndiprops.jar` file in a suitable directory; for example,*$LIBERTY_HOME*/usr/shared/resources/lib/jndiprops.jar.

Edit the *$LIBERTY_HOME*/usr/servers/worklightServer/server.xml file. For each `<application>` element, add or update the `<classloader>` element.

```
<application ...>
    ...
    <classloader delegation="parentLast"></p>
        ...
        <privateLibrary>
            <fileset dir="${shared.resource.dir}/lib"
               includes="jndiprops.jar"/>
        </privateLibrary>
    </classloader>
</application>
```

Restart the web application server after these changes.

Apache Tomcat

Place the shared library, `jndiprops.jar` file, in a suitable directory; for example, *$TOMCAT_HOME*/`Worklight/jndiprops.jar`.

Edit the *$TOMCAT_HOME*/`conf/server.xml` file. For each <Context> element, add or update the <Loader> element.

```
<Context docBase="worklightconsole" path="/worklightconsole">
    <Loader className="org.apache.catalina.loader.VirtualWebappLoader"
       virtualClasspath="${catalina.base}/Worklight/jndiprops.jar"
       searchVirtualFirst="true"/>
    ...
</Context>
```

For the MobileFirst project, which uses additional shared libraries, the example code is:

```
<Context docBase="worklightconsole" path="/worklight">
    <Loader className="org.apache.catalina.loader.VirtualWebappLoader"
       virtualClasspath="${catalina.base}/Worklight/worklight/worklight-jee-library.jar;${
       searchVirtualFirst="true"/>
    ...
</Context>
```

Restart the web application server after these changes.

Selecting the configuration in a shared library of JNDI configurations

The default configuration is called `default.properties`. If the configuration of JNDI properties has a different name, you must define the **`ibm.worklight.jndi.configuration`** property. The value of this property must be the configuration name without the extension `.properties`. This property can be specified as a Java Virtual Machine system property or explicitly as a JNDI property. This property cannot be defined in the configuration property file.

WebSphere Application Server full profile

You must log in to the WebSphere Application Server administration console and add the setting of the **`ibm.worklight.jndi.configuration`** property to the Java Virtual Machine custom properties.

To add this property setting:

1. Select **Servers** > **Server types** > **WebSphere Application Servers**.
2. Select the name of your server.
3. Expand "Java and process management" and select "Process Definition"
4. Select "Java Virtual Machine".
5. Select "Custom Properties" and click **New**.
6. Specify the name as **`ibm.worklight.jndi.configuration`**.
7. Specify the value as the name of the configuration.
8. Click **Apply**.
9. Click **Save**.

When the property is set, to enable the configuration, restart the appropriate MobileFirst Server components. These components are displayed in the WebSphere Application Server administration console under WebSphere enterprise applications.

WebSphere Application Server Liberty profile

You must edit the *$LIBERTY_HOME*/`usr/servers/worklightServer/`

bootstrap.properties file and set the **ibm.worklight.jndi.configuration** property to point to the name of the configuration.

For example:

```
ibm.worklight.jndi.configuration=testconf
```

Alternatively, instead of editing the bootstrap.properties file, create or edit the $LIBERTY_HOME/usr/servers/worklightServer/jvm.options file. For example, add:

```
-Dibm.ibm.worklight.jndi.configuration=testconf
```

To enable the configuration, restart the web application server.

Apache Tomcat

You must edit the *$TOMCAT_HOME*/conf/catalina.properties file and set the **ibm.worklight.jndi.configuration** property to point to the name of the configuration. For example:

```
ibm.worklight.jndi.configuration=testconf
```

Alternatively, instead of editing the catalina.properties file, depending on the operating system, create or edit one of the following files:

- On UNIX systems: *$TOMCAT_HOME*/bin/setenv.sh

  For example, add:

  ```
  CATALINA_OPTS="$CATALINA_OPTS -Dibm.worklight.jndi.configuration=testconf"
  ```

- On Microsoft Windows systems: *$TOMCAT_HOME*/bin/setenv.bat

  For example, add:

  ```
  set CATALINA_OPTS=%CATALINA_OPTS% -Dibm.worklight.jndi.configuration=testconf
  ```

To enable the configuration, restart the web application server.

Adding the shared library (WebSphere Application Server full profile)

Define the shared library and specify which web applications use it.

Before you begin

Install the shared library jndiprops.jar in a suitable directory in the WebSphere Application Server installation directory.

About this task

You can install the property file as a shared library for all MobileFirst Server components. To do so, you must log in to the WebSphere Application Server administration console to add the shared library.

Procedure

1. Select **Environment** > **Shared Libraries**.
2. Select your scope in the fields **Node=** and **Server=**.
3. Click **New**.
4. Enter a name, for example, “MobileFirst JNDI Properties”.
5. Enter a description, for example, “IBM MobileFirst JNDI property package”.
6. Enter the classpath of the jndiprops.jar file. The *$WAS_INSTALL_DIR*/profiles/*profile-name* directory can be specified as ${USER_INSTALL_ROOT}.
7. Select the option “Use an isolated class loader for this shared library”.

8. Click **Apply**.
9. Click **Save**.
10. Specify which web applications should use the shared library.
    a. In the administration console, select **Applications** > **Application Types** > **WebSphere enterprise applications**. You should stop all applications that you are going to change, because the operations run faster when the applications are stopped.
    b. Select an application, for example, IBM MobileFirst Administration Service.
    c. Select **Shared library references**.
    d. In "Application", select IBM MobileFirst Administration Service.
    e. Click **Reference shared libraries**.
    f. Move the MobileFirst JNDI Properties library from Available to Selected.
    g. Click **OK**.
    h. Click **Save**.

    Repeat this procedure for the other required web applications from among MobileFirst Operations Console, MobileFirst project, Application Center Service, Application Center Console.

What to do next

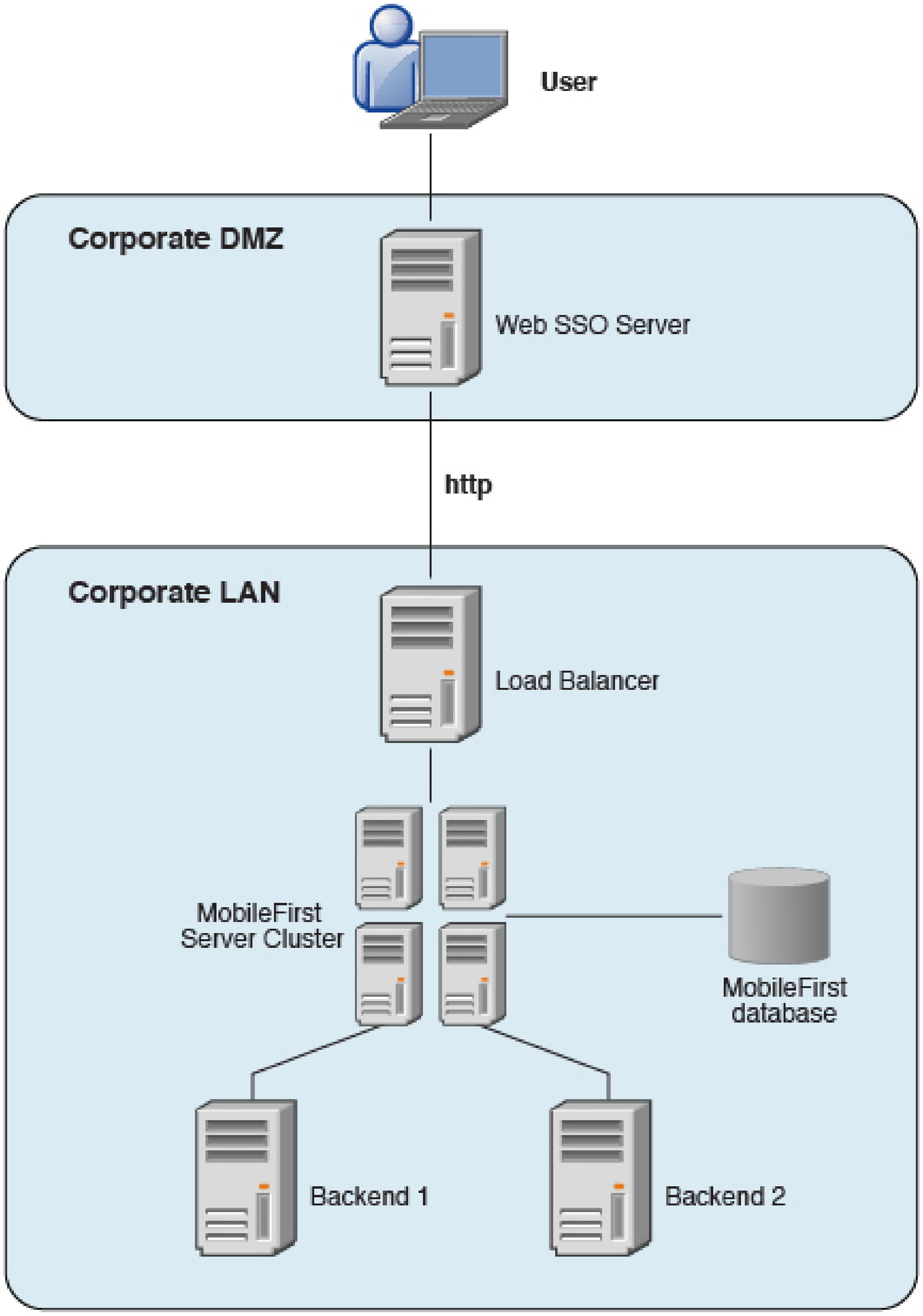
Go to **Applications** > **Application Types** > **WebSphere enterprise applications** and restart all the web applications.

Typical topologies of a MobileFirst instance in an extranet infrastructure

A MobileFirst instance uses a particular topology that is typical for organizations with an established extranet infrastructure.

The following figure depicts this topology.

Figure 6-26. Typical topology of a MobileFirst instance



Such a topology is based on the following principles:

- MobileFirst Server is installed in the organization local area network (LAN), connecting to various enterprise back-end systems.
- MobileFirst Server can be clustered for high availability and scalability.
- MobileFirst Server uses a database for storing push notification information, statistics for reporting and analytics, and the metadata that the server needs at run time. All instances of MobileFirst Server share a single instance of the database.

- MobileFirst Server is installed behind a web Single Sign-On (SSO) authentication infrastructure, which acts as a reverse proxy and provides the Security Socket Layer (SSL).

MobileFirst Server can be installed in different network configurations, which might include several DMZ layers (firewall configurations for securing local area networks), reverse proxies, Network Address Translation (NAT) devices, firewalls, high availability components such as load balancers, IP sprayers, clustering, and alike. Some of these components are explained. However, this document assumes a simpler configuration in which MobileFirst Server is installed in the DMZ.

Related information

- "Integration and authentication with a reverse proxy" on page 14-3
- "Integration with IBM WebSphere DataPower" on page 14-11

Setting up IBM MobileFirst Platform Foundation for iOS in WebSphere Application Server cluster environment

You can set up a MobileFirst cluster environment with IBM WebSphere Application Server Network Deployment V8.5 and IBM HTTP Server.

About this task

This procedure explains how to set up IBM MobileFirst Platform Foundation for iOS in the topology shown in Figure 1:

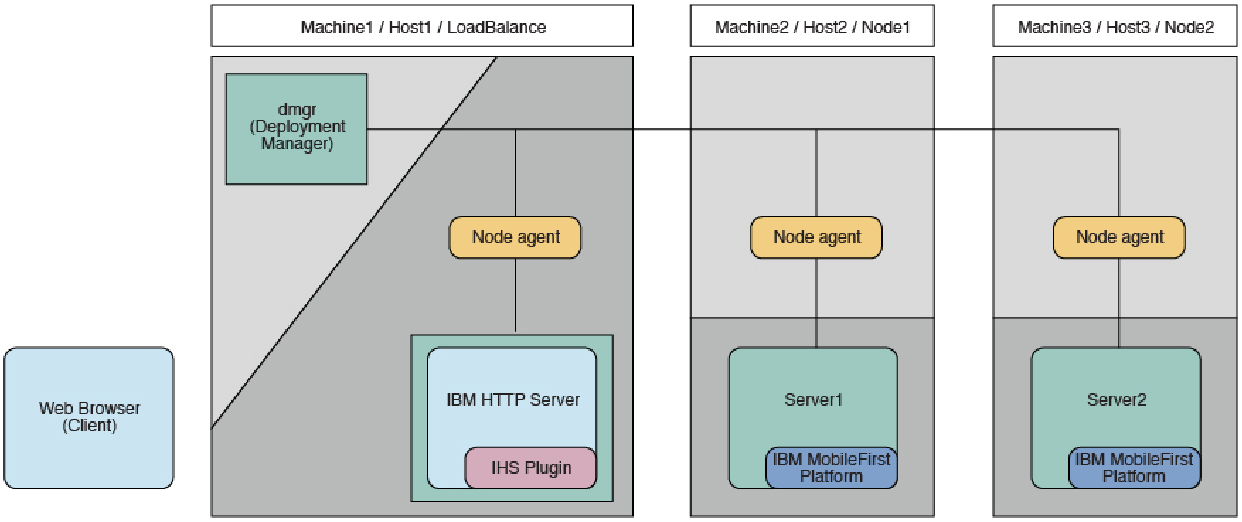


Figure 6-27. MobileFirst cluster topology with IBM WebSphere Application Server Network Deployment

If you install IBM HTTP Server, the administration components require certain JNDI properties to be configured. For more information, see "Using a reverse proxy with server farm and WebSphere Application Server Network Deployment topologies" on page 6-18 and "Defining the endpoint of the MobileFirst Administration services" on page 6-91.

The instructions are based on the hardware and software that are listed in the following Table 6-59 on page 6-302 and Table 6-60 on page 6-302 tables.

Table 6-59. 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 6-60. 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 MobileFirst Platform Foundation for iOS.
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 you 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 you start. It is also included in the WebSphere Application Server installation repository.
IBM MobileFirst Platform Foundation for iOS V7.1.0	IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS installation.
Ant 1.8.3	Configure IBM MobileFirst Platform Foundation for iOS 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 -enableAdminSecurity true
-adminUserName wasadmin -adminPassword wasadmin
-serverType DEPLOYMENT_MANAGER
```

On Linux:

```
./manageprofiles.sh -create -profileName dmgr
-profilePath ../profiles/dmgr506 -templatePath
../profileTemplates/management -enableAdminSecurity true
-adminUserName wasadmin -adminPassword wasadmin
-serverType DEPLOYMENT_MANAGER
```

2) Create an IBM HTTP Server node profile. The following command creates a profile named "ihs":

On Windows:

```
./manageprofiles.bat -create -profileName ihs
-profilePath ../profiles/ihs -templatePath
../profileTemplates/managed
```

On Linux:

```
./manageprofiles.sh -create -profileName ihs -profilePath
../profiles/ihs506 -templatePath ../profileTemplates/
managed
```

3) Start the deployment manager:

On Windows:

```
./startManager.bat
```

On Linux:

```
./startManager.sh
```

4) Add an IBM HTTP Server node to the deployment manager. The following command adds the node defined by the “ihs” profile to the deployment manager running on Host1, and assigns port 8879:

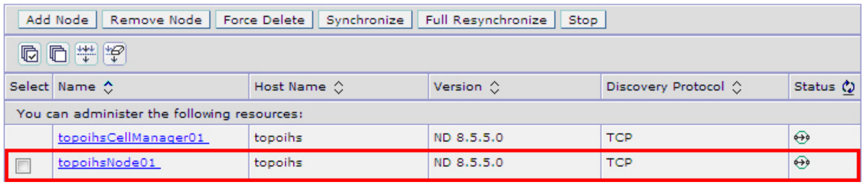
   On Windows:

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

   On Linux:

   ```
   ./addNode.sh Host1 8879 -profileName ihs
   ```

5) From the WebSphere Application Server administrative console, click **System administration** > **Nodes** and check that the node is added to the deployment manager.



Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	topoihsCellManager01	topoihs	ND 8.5.5.0	TCP	⊕
☐	topoihsNode01	topoihs	ND 8.5.5.0	TCP	⊕

   Note: Node names might be different from the profile names you specify because WebSphere Application Server automatically generates a display name for a new node.

c. Create MobileFirst 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:

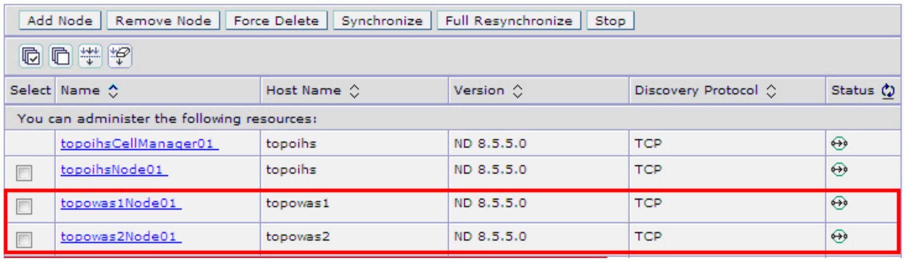
      ```
      ./addNode.bat Host1 8879 -profileName node1
      ```

      On Linux:

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

d. Repeat step 2c to create MobileFirst node2 on Host3.

e. From the WebSphere Application Server administrative console, click **System administration** > **Nodes** and check that the nodes you added to the deployment manager are listed.

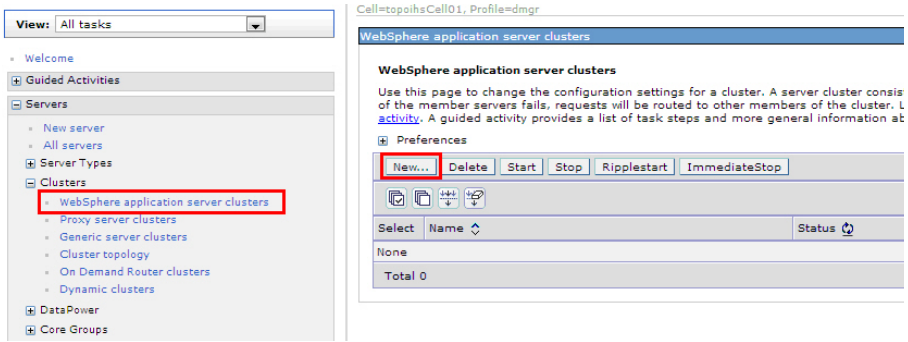


Select	Name	Host Name	Version	Discovery Protocol	Status
You can administer the following resources:					
	topoihsCellManager01	topoihs	ND 8.5.5.0	TCP	⇔
□	topoihsNode01	topoihs	ND 8.5.5.0	TCP	⇔
□	topowas1Node01	topowas1	ND 8.5.5.0	TCP	⇔
□	topowas2Node01	topowas2	ND 8.5.5.0	TCP	⇔

Note: Node names might be different from the profile names you specify because WebSphere Application Server automatically generates a display name for a new node.

3. Create a cluster and add MobileFirst 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.

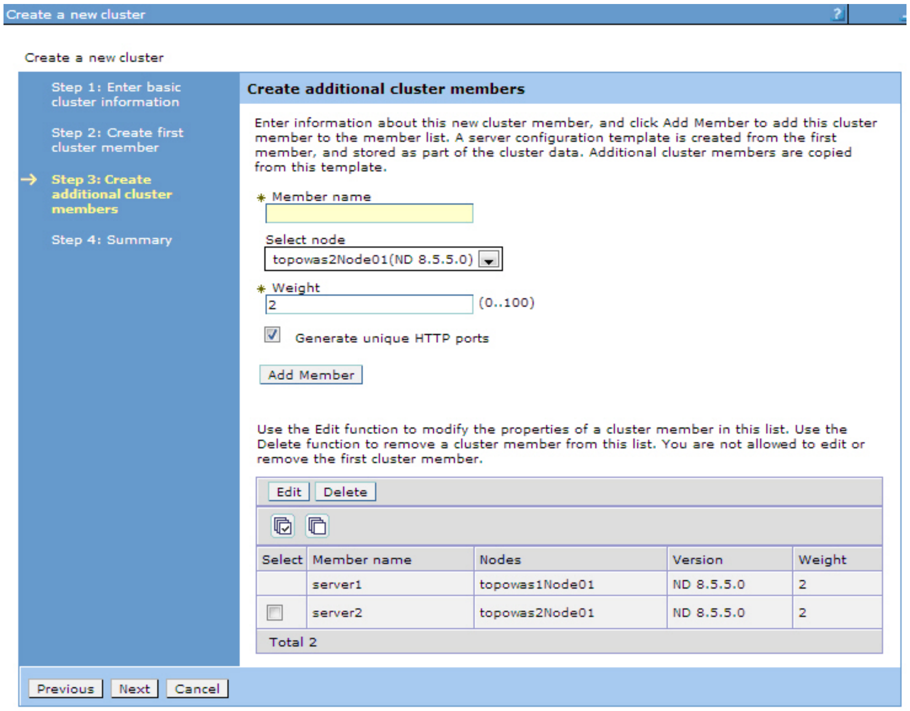


WebSphere application server clusters

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

Select	Name	Status
None		
Total 0		

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


Create a new cluster
Create a new cluster
Step 1: Enter basic cluster information
Step 2: Create first cluster member
Step 3: Create additional cluster members
Step 4: Summary

Create additional cluster members

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

* Member name

Select node
topowas2Node01(ND 8.5.5.0)

* Weight
2 (0..100)

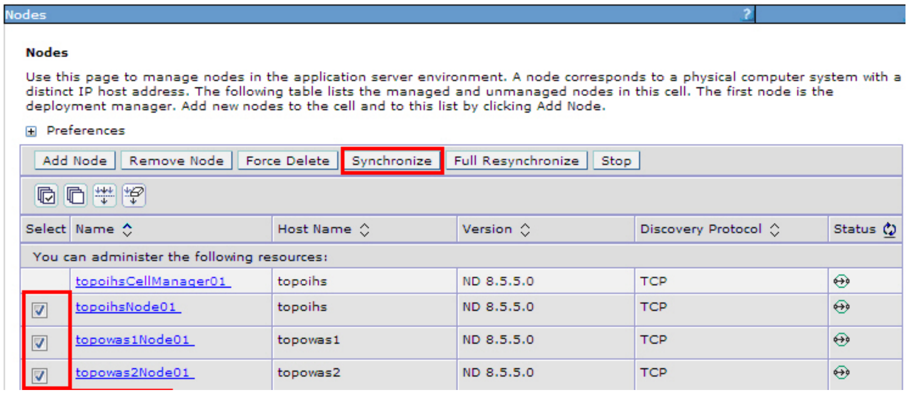
☑ Generate unique HTTP ports

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

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

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

d. If the status column indicates that nodes are not synchronized, click **System Administration** > **Nodes**, and then click **Synchronize** to synchronize your nodes to the deployment manager.



Nodes

Nodes

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

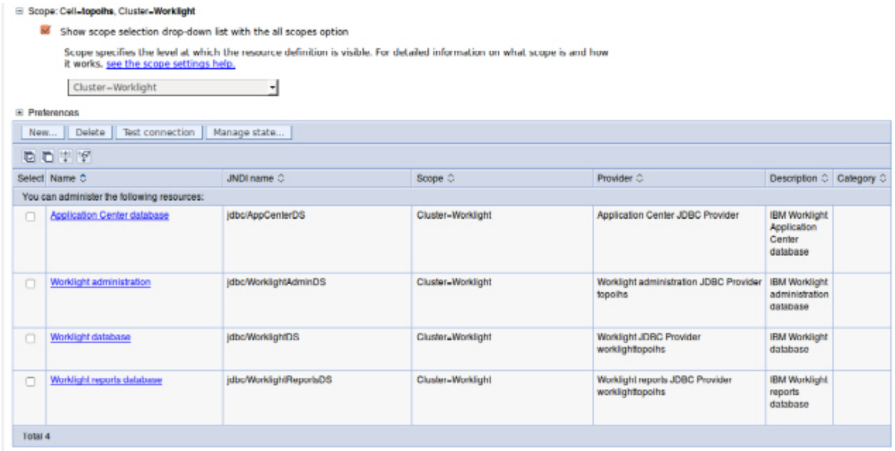
⊞ Preferences

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

4. Install MobileFirst 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 MobileFirst Server” on page 6-3.
5. Configure the databases. For instructions, see “Creating and configuring the databases with Ant tasks” on page 11-15.
6. In IBM MobileFirst Platform Command Line Interface for iOS, create a project and build a MobileFirst project WAR file. See “Artifacts produced during development cycle” on page 8-22.
7. Configure IBM MobileFirst Platform Foundation for iOS 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 11-16. 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.



⊟ Scope: Cell=topoihs, Cluster=Worklight

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

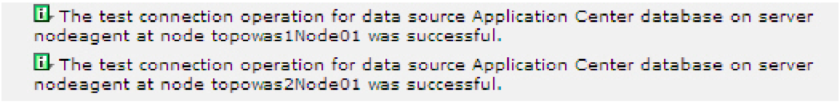
Scope specifies the level at which the resource definition is visible. For detailed information on what scope is and how it works, see the scope settings help.

Cluster=Worklight

⊞ Preferences

Select	Name	JNDI name	Scope	Provider	Description	Category
You can administer the following resources:						
□	Application Center database	jdbc/AppCenterDS	Cluster=Worklight	Application Center JDBC Provider	IBM Worklight Application Center database	
□	Worklight administration	jdbc/WorklightAdminDS	Cluster=Worklight	Worklight administration JDBC Provider topoihs	IBM Worklight administration database	
□	Worklight database	jdbc/WorklightDS	Cluster=Worklight	Worklight JDBC Provider worklightopoihs	IBM Worklight database	
□	Worklight reports database	jdbc/WorklightReportsDS	Cluster=Worklight	Worklight reports JDBC Provider worklightopoihs	IBM Worklight reports database	
Total 4						

c. Select the 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 MobileFirst Operations Console application is running.

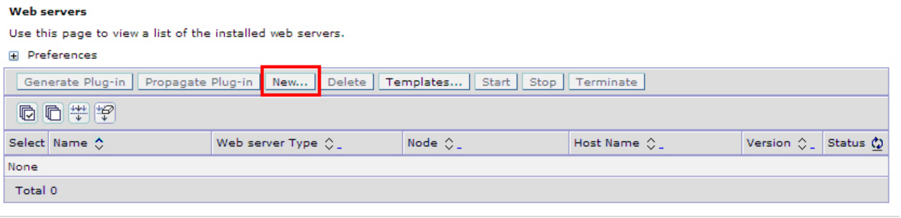
e. Now that you have deployed IBM MobileFirst Platform Foundation for iOS on the two node servers, you can access the MobileFirst Operations Console on each host by browsing to the associated URLs:

- `http://Host2:9080/worklightconsole`
- `http://Host3:9080/worklightconsole`

Check that both MobileFirst Operations 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.



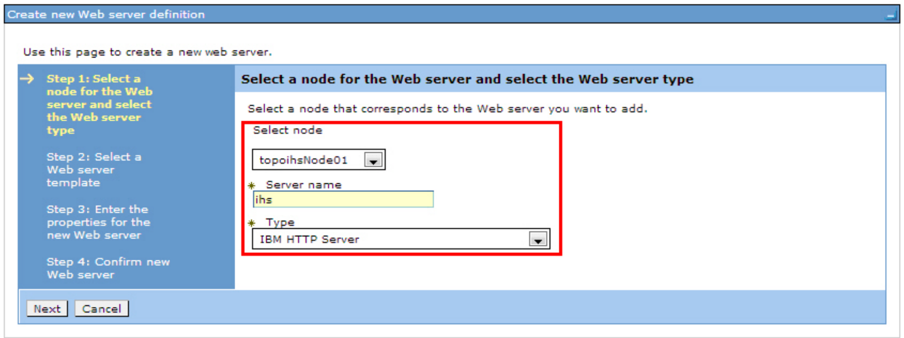
Web servers

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

⊞ Preferences

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

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



Create new Web server definition

Use this page to create a new web server.

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

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

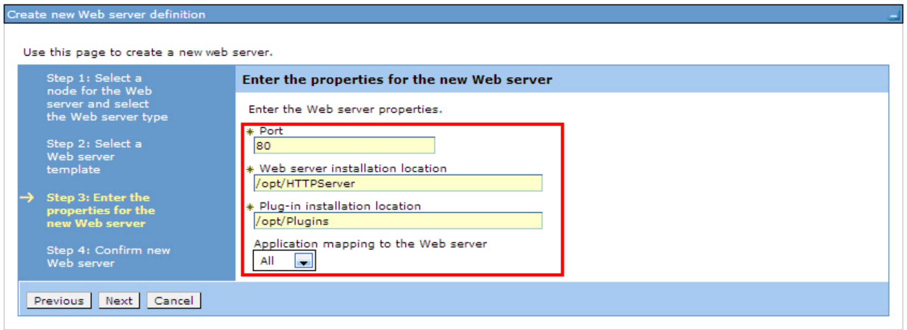
Select node: topoihsNode01

* Server name: ihs

* Type: IBM HTTP Server

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

c. Enter the IBM HTTP Server home and Web Server Plugins home you previously selected on Host1, and then click **Next** and save your configuration.



Create new Web server definition

Use this page to create a new web server.

Enter the properties for the new Web server

Enter the Web server properties.

* Port: 80

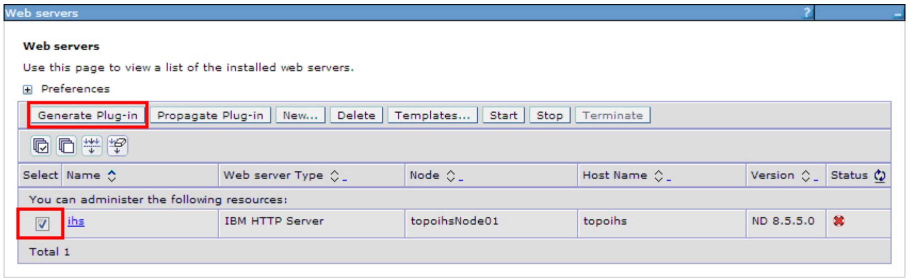
* Web server installation location: /opt/HTTPServer

* Plug-in installation location: /opt/Plugins

Application mapping to the Web server: All

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

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



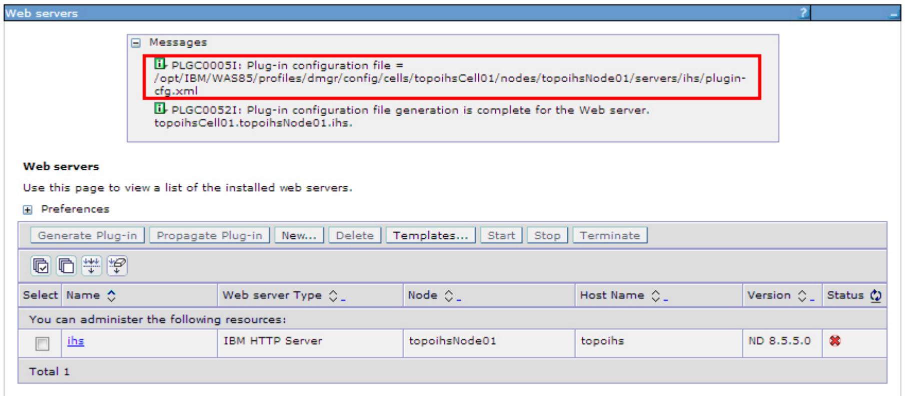
Web servers

Web servers

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

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

A confirmation message is displayed.



Web servers

Messages

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

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

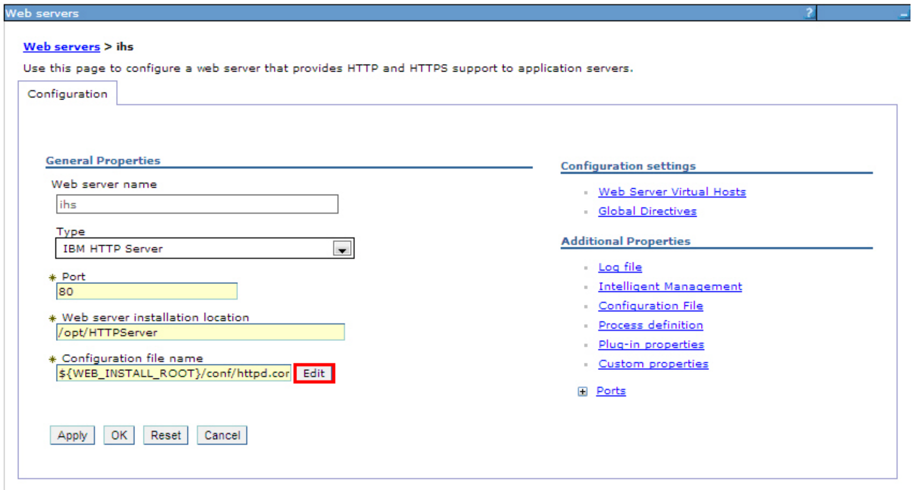
Web servers

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

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

e. Make a note of the `plugin-cfg.xml` location displayed in the confirmation message.

f. In the administrative console, on the Web servers page, click **ihs**, and then in the **Configuration file name** field, click **Edit**.



Web servers > ihs

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

Web server name: ihs

Type: IBM HTTP Server

* Port: 80

* Web server installation location: /opt/HTTPServer

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

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

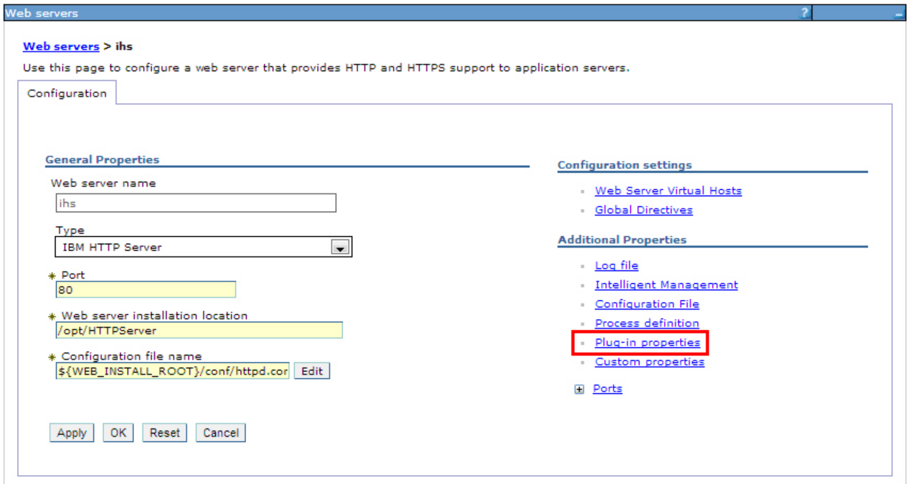
On Windows:

```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.dll
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

On Linux:

```
LoadModule was_ap22_module {IHS_Plugin_HOME}/bin/{64bits}/mod_was_ap22_http.so
WebSpherePluginConfig {path to}/plugin-cfg.xml
```

h. In the administrative console, on the Web servers page for the "ihs" server, click **Plug-in properties**.



Web servers > ihs

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

Web server name: ihs

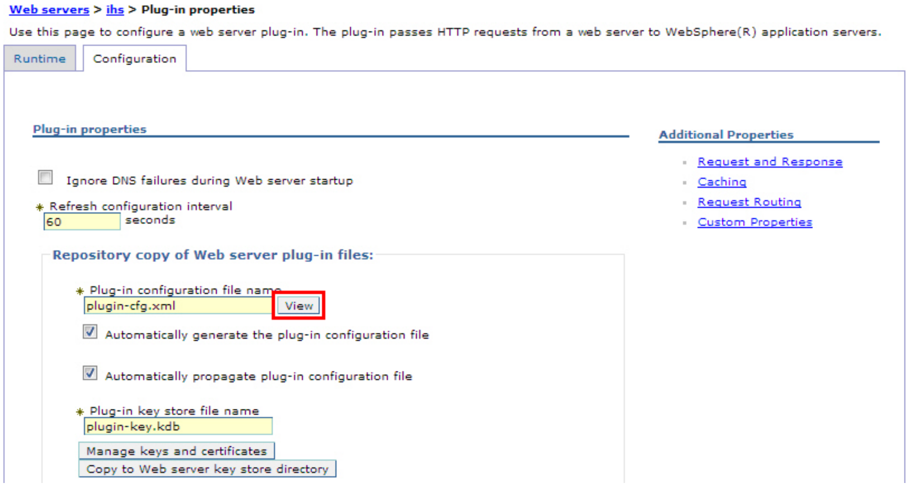
Type: IBM HTTP Server

* Port: 80

* Web server installation location: /opt/HTTPServer

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

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



Web servers > ihs > Plug-in properties

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

Plug-in properties

□ Ignore DNS failures during Web server startup

* Refresh configuration interval
60 seconds

Repository copy of Web server plug-in files:

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

☑ Automatically generate the plug-in configuration file

☑ Automatically propagate plug-in configuration file

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

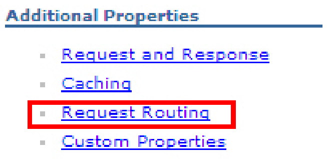
j. Search for the cluster node and MobileFirst URI in the `plugin-cfg.xml` file. For example:

```
<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>
```

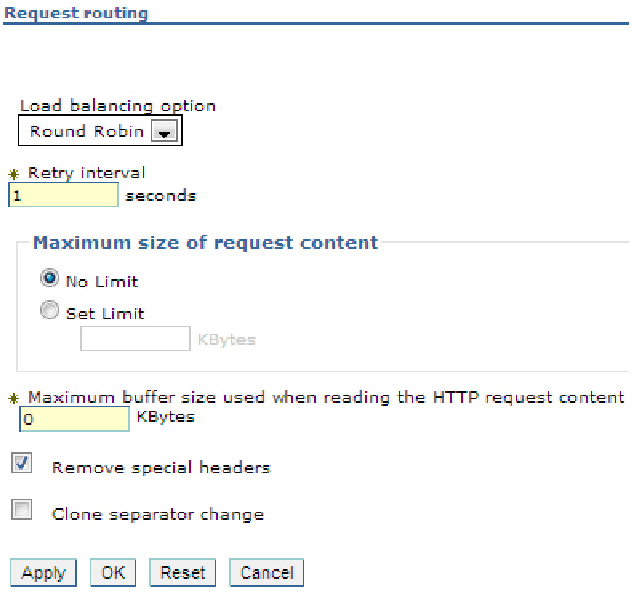
If your configuration file does not include cluster servers and URIs, delete the "ihs" server and create it again.

k. Optional: On the Plug-in properties page for the "ihs" server, click **Request Routing** if you want to set a load-balancing policy.



Additional Properties

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



Request routing

Load balancing option: Round Robin

* Retry interval: 1 seconds

Maximum size of request content

● No Limit

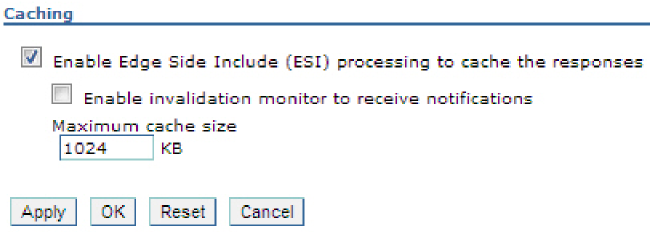
○ Set Limit ____ KBytes

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

☑ Remove special headers

□ Clone separator change

l. Optional: On the Plug-in properties page for the "ihs" server, click **Caching** if you want to configure caching.



Caching

☑ Enable Edge Side Include (ESI) processing to cache the responses

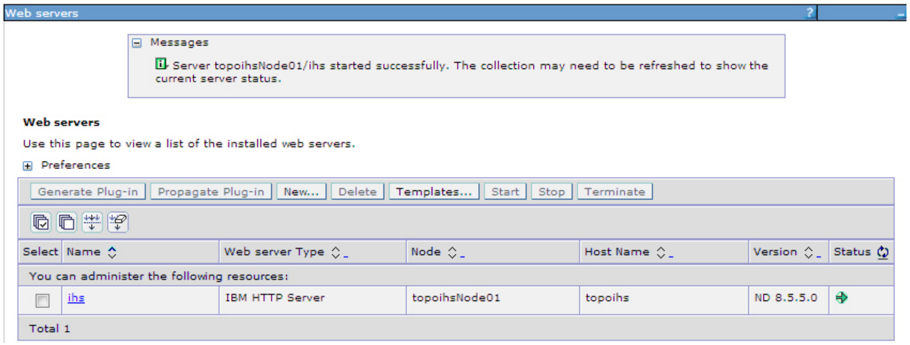
□ Enable invalidation monitor to receive notifications

Maximum cache size: 1024 KB

10. Start the IBM HTTP server and verify that the server is running.

   a. In the WebSphere Application Server administrative console, go to **Servers** > **Server Types** > **Web servers**.

   b. Select the IBM HTTP server you created (in this example, named "ihs"), and then click **Start**.



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

Web servers

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

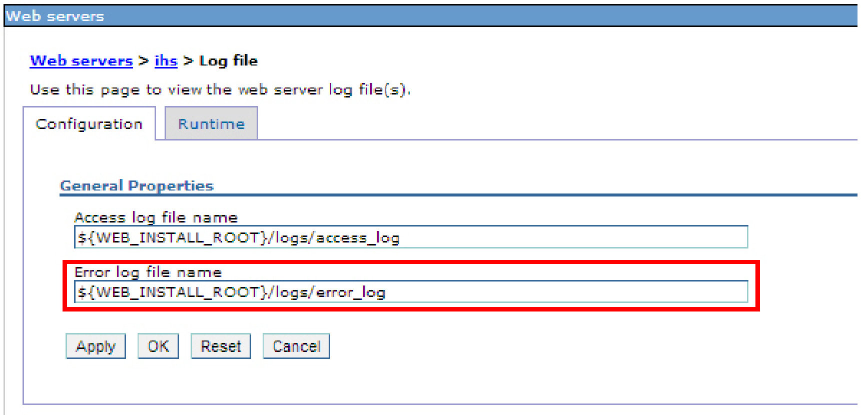
Select	Name	Web server Type	Node	Host Name	Version	Status
You can administer the following resources:						
□	ihs	IBM HTTP Server	topoihsNode01	topoihs	ND 8.5.5.0	➔
Total 1						

c. If the server fails to start, check the log file. To find the location of the log file:

1) In the administrative console, on the Web servers page for the "ihs" server, click **Log file**.

2) On the log file page, click the **Configuration** tab.

3) The location of the log file is displayed in the **Error log file name** field.



Web servers > ihs > Log file

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

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

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

d. To verify that the IBM HTTP server is running, enter the URL for the MobileFirst Operations Console in a web browser. For example: `http://Host1:80/worklightconsole`.

Results

IBM MobileFirst Platform Foundation for iOS is now installed on an IBM WebSphere Application Server Network Deployment cluster, and is ready for use.

Setting up HTTP Server in a WebSphere Application Server Liberty profile farm

You can set up an IBM HTTP Server in a MobileFirst cluster environment with the Liberty profile.

Before you begin

Install a server farm for Liberty. See “Installing a server farm” on page 6-127. If a server farm is not configured, the MobileFirst Server installation does not work properly. As a consequence, the changes that are made by using the MobileFirst Operations Console or the Administration Service are not replicated to all the servers of the farm. The result is inconsistent behavior for client devices that connect to the MobileFirst Server.

About this task

You can set up IBM MobileFirst Platform Foundation for iOS in the topology similar to the one shown in Figure 6-28.

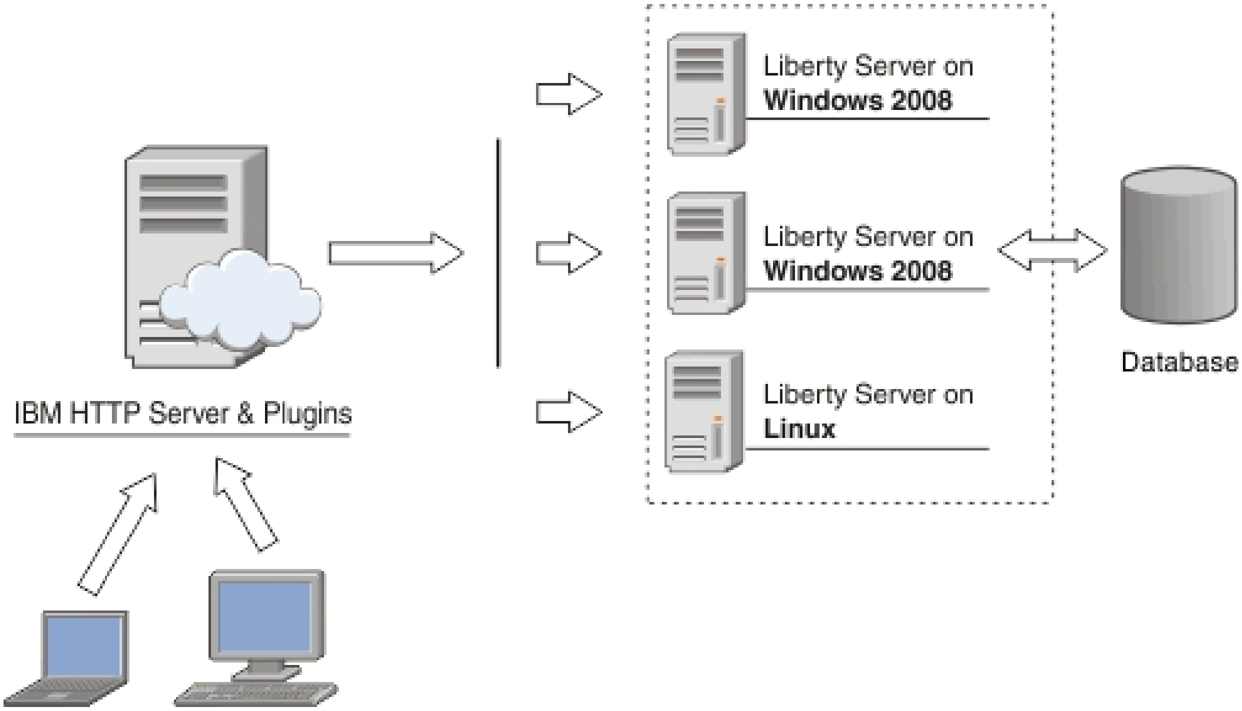


Figure 6-28. MobileFirst cluster topology with the Liberty profile

If you install IBM HTTP Server, you set up some JNDI properties to configure the administration components. For more information, see "Using a reverse proxy with server farm and WebSphere Application Server Network Deployment topologies" on page 6-18 and "Defining the endpoint of the MobileFirst Administration services" on page 6-91.

This procedure uses the following hardware and software elements.

Table 6-61. Hardware

Host name	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 6-62. Software

Name	Description
IBM Installation Manager	Install IBM HTTP Server , IBM Liberty profile, and IBM MobileFirst Platform Foundation for iOS.

Table 6-62. Software (continued)

Name	Description
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.
Web Server Plug-ins	You need to get access to the installation repository before you start the procedure. IBM HTTP Server Plug-in 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 MobileFirst Platform Foundation for iOS	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 MobileFirst Platform Foundation for iOS installation.

Procedure

1. Install IBM HTTP Server and Web Server Plug-ins.
   a. On the Host1 server, log on as the `root` user ID and run IBM Installation Manager to install the IBM HTTP server and Web Server Plug-ins. This documentation assumes that the applications are installed in the following places:

      IBM HTTP Server home
      `/opt/HTTPServer`

      Web Server Plug-ins home
      `/opt/Plugins`
2. Start the Liberty profile servers to test whether you can access the MobileFirst Operations Console on Host2 and Host3 by browsing to the associated URLs:
   - `http://Host2:9080/worklight/console`
   - `http://Host3:9080/worklight/console`

   Check that both MobileFirst Operations Console are running.
3. Run the following command on Host1 to start the IBM HTTP server.

   ```
   /opt/HTTPServer/bin/apachectl start
   ```

   If you encounter problems when IBM HTTP server is starting, see "Troubleshooting IBM HTTP Server startup" on page 6-319.
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>
```

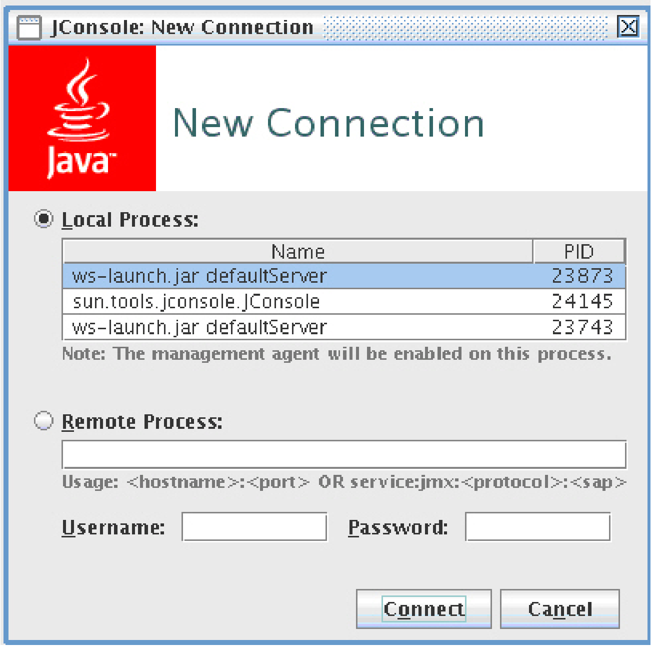
b. Use one of the following methods to generate the `plugin-cfg.xml` file for the Liberty server that is running your application.

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**.

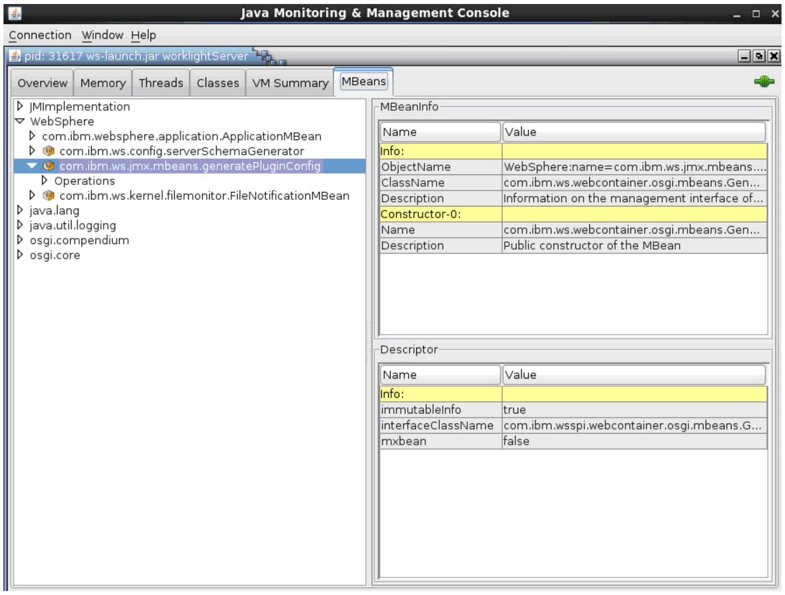


Name	PID
ws-launch.jar defaultServer	23873
sun.tools.jconsole.JConsole	24145
ws-launch.jar defaultServer	23743

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

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

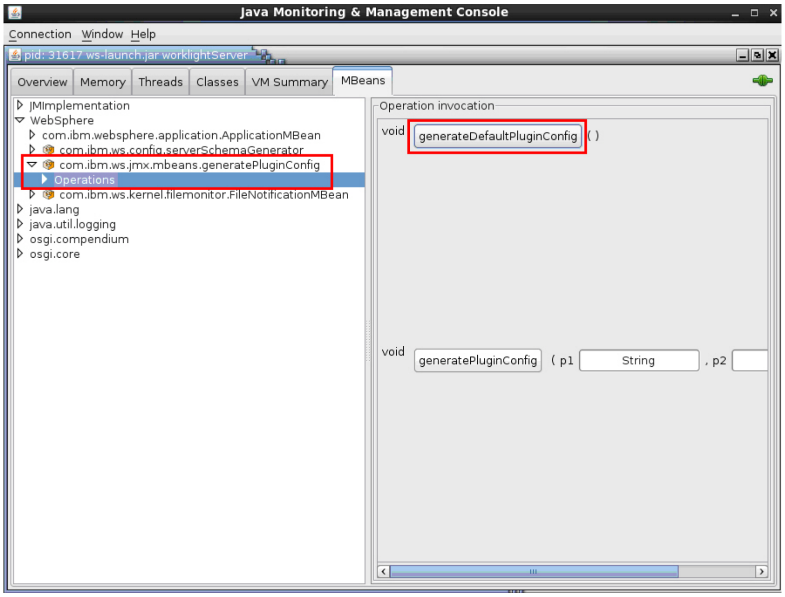
3) In the Java Monitoring & Management Console, click the **MBeans** tab.


Java Monitoring & Management Console
Connection Window Help
pid: 31617 ws-launch.jar worklightServer
Overview
Memory
Threads
Classes
VM Summary
MBeans
JMImplementation
WebSphere
com.ibm.websphere.application.ApplicationMBean
com.ibm.ws.config.serverSchemaGenerator
com.ibm.ws.jmx.mbeans.generatePluginConfig
Operations
com.ibm.ws.kernel.filemonitor.FileNotificationMBean
java.lang
java.util.logging
osgi.compendium
osgi.core
MBeanInfo
Descriptor

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

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

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


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

You can find the generated file in the `\wlp\usr\servers\`*`<server_name>`* directory. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Config ASDisableNagle="false"
  AcceptAllContent="false"
  AppServerPortPreference="HostHeader"
  ChunkedResponse="false"
  FIPSEnable="false"
  IISDisableNagle="false"
  IISPluginPriority="High"
  IgnoreDNSFailures="false"
  RefreshInterval="60"
  ResponseChunkSize="64"
  SSLConsolidate="false"
  SSLPKCSDriver="REPLACE"
  SSLPKCSPassword="REPLACE"
  TrustedProxyEnable="false"
  VHostMatchingCompat="false">
  <Log LogLevel="Error" Name="String\logs\String\http_plugin.log"/>
  <Property Name="ESIEnable" Value="true"/>
  <Property Name="ESIMaxCacheSize" Value="1024"/>
  <Property Name="ESIInvalidationMonitor" Value="false"/>
  <Property Name="ESIEnableToPassCookies" Value="false"/>
  <Property Name="PluginInstallRoot" Value="String"/>
  <VirtualHostGroup Name="default_host">
    <VirtualHost Name="*:443"/>
    <VirtualHost Name="*:80"/>
    <VirtualHost Name="*:9080"/>
  </VirtualHostGroup>
  <ServerCluster CloneSeparatorChange="false"
    GetDWLMTable="false"
    IgnoreAffinityRequests="true"
    LoadBalance="Round Robin"
    Name="String_default_node_Cluster"
    PostBufferSize="64"
    PostSizeLimit="-1"
    RemoveSpecialHeaders="true"
    RetryInterval="60">
    <Server CloneID="s56"
      ConnectTimeout="0"
      ExtendedHandshake="false"
      MaxConnections="-1"
      Name="default_node_String0"
      ServerIOTimeout="900"
      WaitForContinue="false">
        <Transport Hostname="wasvm56" Port="9080" Protocol="http"/>
    </Server>
    <PrimaryServers>
      <Server Name="default_node_String0"/>
    </PrimaryServers>
  </ServerCluster>
  <UriGroup Name="default_host_String_default_node_Cluster_URIs">
    <Uri AffinityCookie="JSESSIONID" AffinityURLIdentifier="jsessionid" Name="/
  </UriGroup>
  <Route ServerCluster="String_default_node_Cluster"
    UriGroup="default_host_String_default_node_Cluster_URIs"
    VirtualHostGroup="default_host"/>
</Config>
```

Eclipse

1) Make sure that 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 by 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 Plug-ins.

   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 Plug-ins. 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** attributes should not be used in a production environment. In the WebSphere Application Server Liberty profile, the **CloneID** attribute is generated when you start a server for the first time; it is regenerated if you start the server with the --clean option.

      In a production use case, manually assigning a clone ID ensures that the **CloneID** attribute is stable and that request affinity is correctly maintained. The **CloneID** attribute must be unique for each server and can be 8 to 9 alphanumeric characters in length.

      The following example shows **CloneID** attributes 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 the round-robin load balancing process is successfully routing application requests to each of the back-end Liberty servers.

Troubleshooting IBM HTTP Server startup

Problems to start the IBM HTTP Server during deployment of a IBM MobileFirst Platform Server on a WebSphere Application Server Liberty profile farm might be caused by an exception in the runtime library.

About this task

When you set up IBM MobileFirst Platform Foundation for iOS 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 caused by an attempt 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 similar message is displayed, you can make the required libraries available as follows.

Procedure

1. Check the IBM HTTP Server libraries:

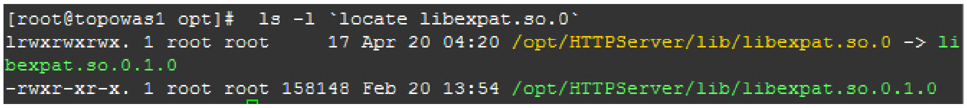
   ```
   ldd /opt/HTTPServer/bin/httpd
   ```

   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 that it includes all `conf` files under `/etc/ld.so.conf.d/`.

```
include ld.so.conf.d/*.conf
```

4. Add the IBM HTTP Server library to the configuration.
   a. `cd /etc/ld.so.conf.d/`
   b. Add the `http` library to the system configuration. The location of the IBM HTTP Server lib is shown in Step 1.
      ```
      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 that `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 instances of MobileFirst Server

You can use IBM WebSphere DataPower as a gateway for all incoming connections for IBM MobileFirst Platform Foundation for iOS and Application Center, and IBM HTTP Server (IHS) for load-balancing MobileFirst 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:

- MobileFirst 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 MobileFirst instance in an extranet infrastructure” on page 6-299.
- IBM MobileFirst Platform Foundation for iOS 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 6-210.
- IBM WebSphere DataPower XI52.
- IBM HTTP Server.
- Any LDAP server with SSL enabled.

About this task

This procedure explains how to set up IBM MobileFirst Platform Foundation for iOS in the topology similar to the one shown in Figure 6-29.

.

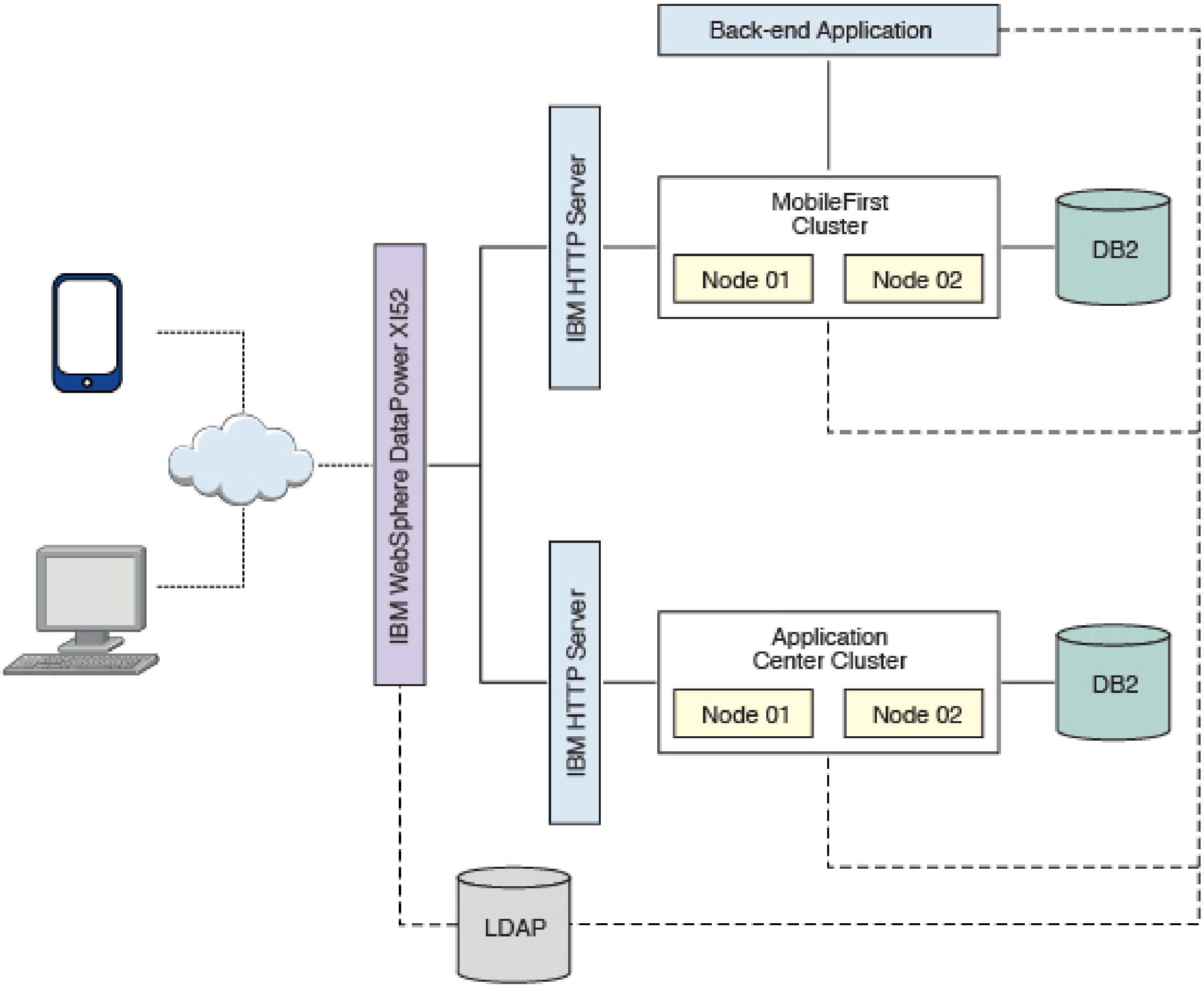


Figure 6-29. IBM MobileFirst Platform Foundation for iOS integration with an IBM WebSphere Application Server 8.5 Cluster or a Liberty profile Server Farm

DataPower XI52 acts as the gateway for all IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Platform Foundation for iOS and Application Center.

The following sequence of events takes place when a mobile application makes a request (see Figure 6-30):

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 MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Platform Foundation for iOS.
6. IBM MobileFirst Platform Foundation for iOS forwards the request to DataPower, and DataPower forwards it to the requesting mobile application.

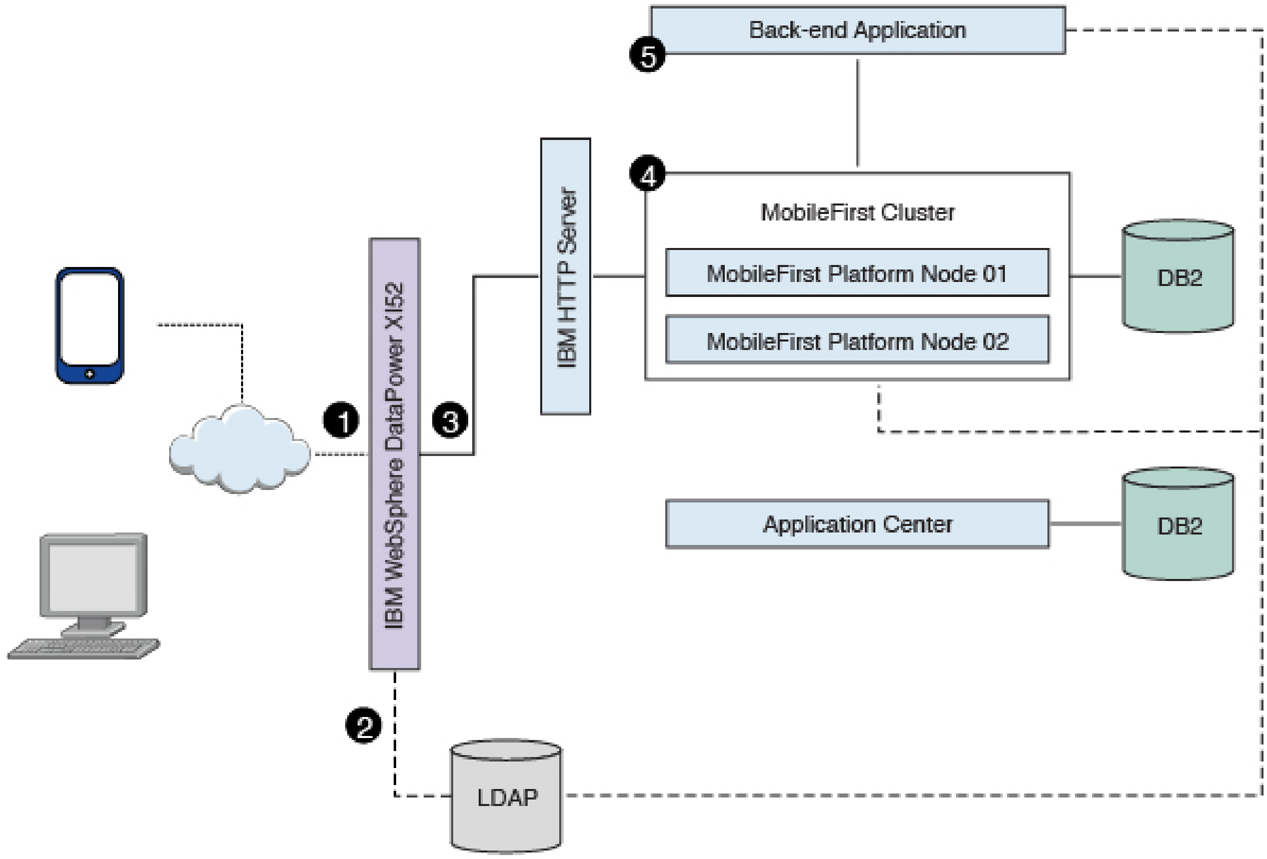


Figure 6-30. Mobile application request-response flow

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 MobileFirst cluster (see Figure 6-31 on page 6-323).

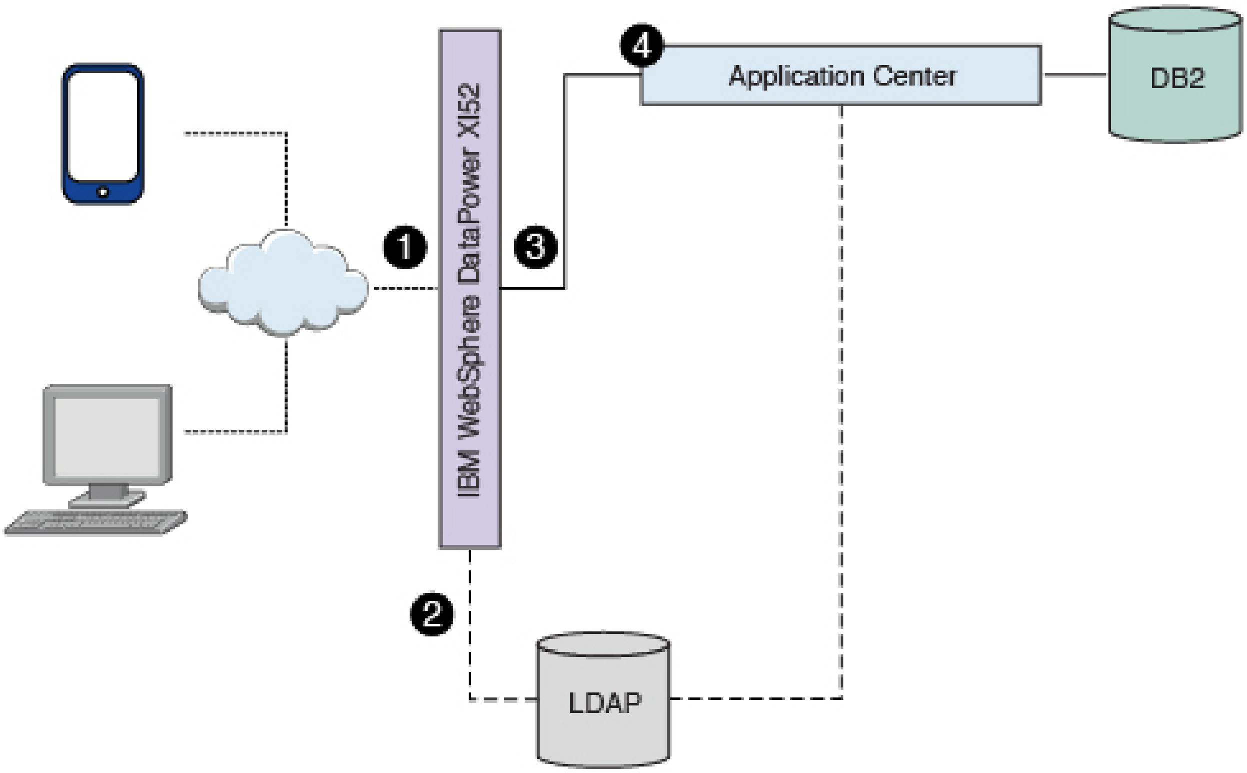


Figure 6-31. Application Center request-response flow

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 MobileFirst 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;groupOfNames:
</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="pa
```

2. Configure MobileFirst Server.

You can secure IBM MobileFirst Platform Foundation for iOS in a typical WebSphere Application Server runtime environment in two ways:

Option 1
Securing WebSphere Application Server using application security and securing the IBM MobileFirst Platform Foundation for iOS WAR file.

Option 2
Securing WebSphere Application Server using application security but not securing the IBM MobileFirst Platform Foundation for iOS 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 11-93 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 MobileFirst 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, MobileFirst Server is composed of one or more runtime environments, an administration console and administration services, an enterprise application store, and an operational analytics feature. MobileFirst Server components run as web applications on an application server. For more information about MobileFirst Server components, see “Introduction to the MobileFirst Server components” on page 6-6.

The roles associated with the MobileFirst Operations Console and Administration Services components are different from the role defined in the WASLTPAModule login module (see “WASLTPAModule login module” on page 8-265). The MobileFirst Operations 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 MobileFirst runtime environments. These roles should be mapped to the users or user groups that have been cleared to access the MobileFirst applications. MobileFirst Operations Console and Administration Services must be set up and configured before you proceed to deploy the MobileFirst runtime services See the following instructions depending on your application server, to map the administration user roles for MobileFirst Operations Console and Administration Services:

- “Configuring WebSphere Application Server full profile for MobileFirst Server administration” on page 6-98
- “Configuring WebSphere Application Server Liberty profile for MobileFirst Server administration” on page 6-99

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 MobileFirst Server and click **Security role to user/group mapping** to map the LDAP user to the MobileFirst roles.

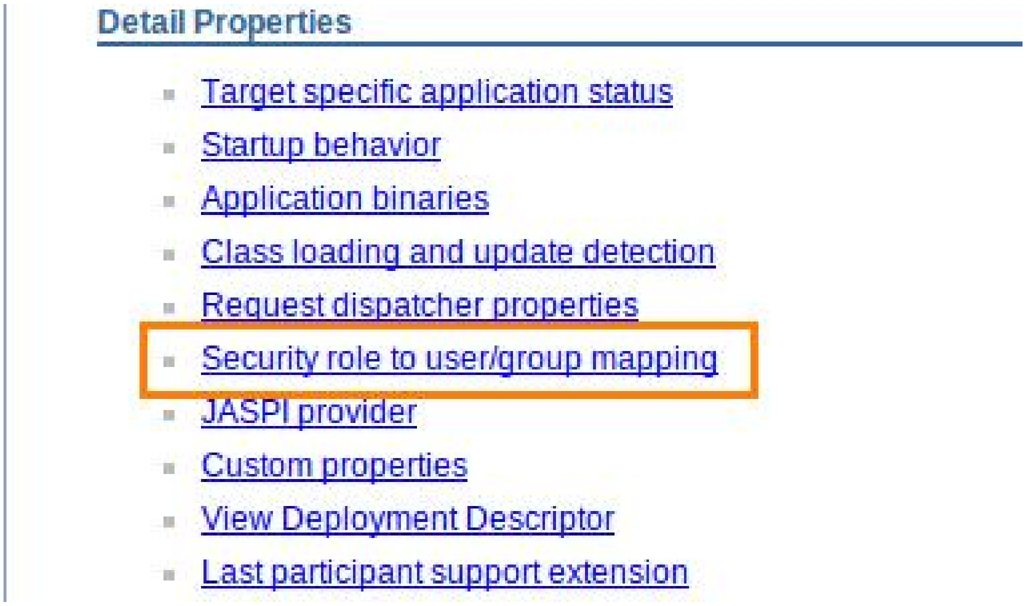

Detail Properties
Target specific application status
Startup behavior
Application binaries
Class loading and update detection
Request dispatcher properties
Security role to user/group mapping
JASPI provider
Custom properties
View Deployment Descriptor
Last participant support extension

Figure 6-32. 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 8-265.

3. Configure Application Center.

   a. Complete the following configuration tasks depending on the server being used:

      - “Configuring the Java EE security roles on WebSphere Application Server full profile” on page 6-244
      - “Configuring the Java EE security roles on WebSphere Application Server Liberty profile” on page 6-246

   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 6-248
- “LDAP with WebSphere Application Server V8.x” on page 6-254
- “LDAP with Liberty profile” on page 6-259

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 application resources (full profile)” on page 6-271
- “Configuring the endpoint of the application resources (Liberty profile)” on page 6-273

4. Configure DataPower. DataPower XI52 acts as the gateway for all IBM MobileFirst Platform Foundation for iOS 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.

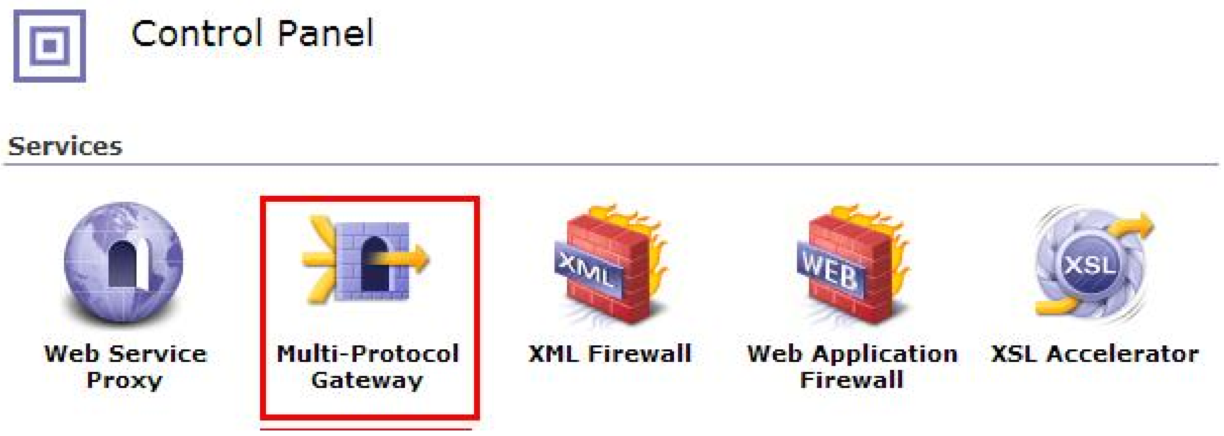


Figure 6-33. Accessing the Multi-Protocol Gateway main page

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 6-63:

Table 6-63. 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 6-64
   2) `worklight-ssl-policy_verifyLTPA`: see Table 6-65
   3) `worklight-ssl-policy_allowSSLLoginPage`: see Table 6-66
   4) `worklight-ssl-policy_worklightSSLLogin`: see Table 6-67 on page 6-330

Table 6-64. 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 6-65. 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 6-332. • Input: INPUT • Output: auth
Result	Not applicable.

Table 6-66. Properties of `worklight-ssl-policy_allowSSLLoginPage`

Property	**Value**
Direction	Client to Server.

Table 6-66. Properties of `worklight-ssl-policy_allowSSLLoginPage` (continued)

Property	Value
Match	- Type = URL - Pattern = `/(Login|Error)Page\.htm(l)?(\?originalUrl=.*)?`
AAA	- Input: INPUT - Output: NULL Add a new AAA Policy named `AllowSSLLoginPage` with the following configuration: - Method: HTML Form-based Authentication - HTML Form Policy: Create one with the default values, but edit these values: – Use SSL For Login: enabled – SSL Port: port on which the multi-protocol gateway is listening. - Authentication: Pass identity token to authorization phase - Resource extraction: URL sent by client - Authorization: Always allow
Result	Not applicable.

Table 6-67. 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 6-67. 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 keys file – LTPA key file password: Specify the password for the LTPA keys file.
Transform	Upload `route.xsl` file. See “Sample dynamic routing stylesheet” on page 6-332. • 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.

Related information

Integrating with IBM WebSphere DataPower
Use IBM WebSphere DataPower as a reverse proxy and security gateway for handling inbound mobile traffic, or as a proxy and gateway for handling outbound traffic to a notification mediator.

Integration and authentication with a reverse proxy
You can use a reverse proxy to enable enterprise connectivity within a MobileFirst environment and to provide authentication to IBM MobileFirst Platform Foundation for iOS.

MobileFirst Security and LTPA
A lightweight third-party authentication (LTPA) token is a type of security token 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 back-end services. It can also be used as a single sign-on (SSO) token between the user and multiple servers.

Sample dynamic routing stylesheet

You can use this sample stylesheet to handle the dynamic routing of requests between IBM MobileFirst Platform Foundation for iOS and IBM MobileFirst Platform 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 instances of MobileFirst Server” on page 6-320, Table 6-65 on page 6-329), and when you define rule `worklight-ssl-policy_worklightSSLLogin` (see “Integrating IBM WebSphere DataPower with a cluster of instances of MobileFirst Server” on page 6-320, Table 6-67 on page 6-330).

```
<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-Control-Request-Headers')"/>
    <xsl:variable name="accessControlRequestMethod" select="dp:http-request-header('Access-Control-Request-Method')"/>

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


      <xsl:when test="contains(dp:variable('var://service/URI'), $applicationcenter)
or contains(dp:variable('var://service/URI'), $appcenterconsole)">
        <dp:set-http-request-header name="'Content-Type'" value="$httpContentType"/>
        <dp:set-http-request-header name="'Access-Control-Request-Headers'" value="$accessControlRequestHeaders"/>
        <dp:set-http-request-header name="'Access-Control-Request-Method'" value="$accessControlRequestMethod"/>
        <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_check'"/>
      </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>
```

Endpoints of the MobileFirst Server production server

You can create whitelists and blacklists for the endpoints of the IBM MobileFirst Platform Server.

Note: Information regarding URLs that are exposed by IBM MobileFirst Platform Foundation for iOS is provided as a guideline. Organizations must ensure the URLs are tested in an enterprise infrastructure, based on what has been enabled for white and black lists.

Table 6-68. MobileFirst applications

API URL, under *<application root context>***/**	**Description**	**Suggested for whitelist?**	**For more information**
apps/services/api/*	Used by client applications for operations such as init, Direct Update requests, invocation of adapter procedures, and more.	Yes	"HTTP Interface of the production server" on page 6-336
apps/services/random/*	Used for generating a random number. Used by JSON store implementation and encrypted cache on the client side.	Yes, if you plan to use offline storage such as JSON store.	"JSONStore overview" on page 8-125
apps/services/reach	Used for the reach API, this servlet returns status 200 with OK, letting you verify that the MobileFirst Server is up and running.	Yes	
apps/services/www/*	Used by mobile web or desktop application to access its resources.	Yes	Web application resource requests
apps/services/download/*	Deprecated	No	

Table 6-68. MobileFirst applications (continued)

API URL, under *<application root context>*/	**Description**	**Suggested for whitelist?**	**For more information**
apps/services/preview/*	Used to preview the application.	No. Used for development and administration purposes.	Preview application resource requests

Table 6-69. Direct update

API URL, under *<application root context>*/	**Description**	**Suggested for whitelist?**	**For more information**
directUpdate/*	Used for serving the direct update `.zip` file.	Yes, if you plan to use Direct Update.	

Table 6-70. Node synchronization

API URL, under *<application root context>*/	**Description**	**Suggested for whitelist?**	**For more information**
node/integration/*	Used to receive notifications from IBM MobileFirst Platform Foundation for iOS adapters that are based on Node.js. Not in use and can be blocked.	No	

Table 6-71. Vitality

API URL, under *<application root context>*/	**Description**	**Suggested for whitelist?**	**For more information**
ws/rest/vitality	Used to check server availability. Returns a list of applications and adapters. For server administrators.	No	"Vitality queries for checking server health" on page 13-2

Table 6-72. Invoke back end procedure

API URL, under *<application root context>***/**	**Description**	**Suggested for whitelist?**	**For more information**
invoke	Used to invoke an adapter procedure.	Yes, if application uses adapter authentication features, or if you want to access the adapter directly and not from the application. **Note:** If this API passes the firewall, everyone is able to invoke any adapter procedure and it is protected only by the adapter security test and not by the application security test.	"Accessing adapters from the `/invoke` endpoint" on page 8-120
subscribeSMS	Push subscription service API. Used by applications.	Yes, if application uses push subscription API.	"Web-based SMS subscription" on page 8-182
receiveSMS	SMS subscription service API. Used by applications.	Yes, if application uses SMS subscription API.	"Using two-way SMS communication" on page 8-186

Table 6-73. Client side logging

API URL, under *<application root context>***/**	**Description**	**Suggested for whitelist?**	**For more information**
apps/services/loguploader/*	Used by client applications to upload their accumulated debug and analytics logs.	Yes	"Client-side log capture" on page 8-298
apps/services/configprofile/*	Used by client applications to GET their log configuration, which the admin set via the Log Configuration tab in the IBM MobileFirst Platform Operations Console.	Yes	"Client-side log capture" on page 8-298

Table 6-74. Dev

API URL, under *<application root context>*/	Description	Suggested for whitelist?	For more information
dev/*	Development service API such as /invoke, /appdata, /preview, and others. Used in development environments only.	No, only for the development environment and not for QA, preproduction, or production.	

Table 6-75. Unstructured Supplementary Service Data (USSD)

API URL, under *<application root context>*/	Description	Suggested for whitelist?	For more information
ussd/*	Used for communication with the USSD gateway.	Yes	"USSD Support" on page 8-102

Table 6-76. OAuth Server

API URL, under *<application root context>*/	Description	Suggested for whitelist?	For more information
authorization/v1/ clients/instance	Used by clients applications to register with the OAuth Server	No	"OAuth-based security model" on page 8-188
authorization/v1/ authorization	Used by client applications to perform authorization	Yes, if you are using OAuth	"OAuth-based security model" on page 8-188
authorization/v1/ token	Used by client applications to obtain access tokens	Yes, if you are using OAuth	"OAuth-based security model" on page 8-188
authorization/v1/ publickey	Used by external resource filters to obtain the public key of the MobileFirst Server	Yes, if you are using OAuth	"OAuth-based security model" on page 8-188
authorization/v1/ token/validation	Used for performing online token validation	Yes, if you are using OAuth	"OAuth-based security model" on page 8-188
authorization/v1/ clients/preview	Used for registering applications in preview mode	No, only for development	"OAuth-based security model" on page 8-188

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 6-77. 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 6-78. 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 MobileFirst Server	
`Application ID`	String	Unique Identifier of the application within the MobileFirst Server. Every application deployed on the MobileFirst 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, desktopbrowser, iOSnative, ipad, iphone, JavaMEnative, mobilewebapp`
`Action`	String	Requested action	Details in following table

Table 6-79. 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.

Table 6-79. Actions (continued)

Action	HTTP Request	Parameters
logactivity	POST	**x, isAjaxRequest** – see the following table showing common parameters. **activity** – string.
query	POST	**x, isAjaxRequest** – see the following table showing common parameters.**filterList** – **JSON** block **parameterList** – **JSON** block **sorterList** – **JSON** block **Note:** When the action is **query**, the request URL has the following structure: `.../query/{Adapter Name}/{Procedure Name}` where **Adapter Name** and **Procedure Name** are strings.
logout	POST	**x, isAjaxRequest** - see the following table showing common parameters.
login	POST	**x, isAjaxRequest** – see the following table showing common parameters. **realm** – string.
updates	POST	**x, isAjaxRequest** – see the following table showing common parameters. **skin** – current skin name (string) **checksum** – the checksum of the current skin (string) **skinLoaderChecksum** – the checksum of the skin selection code (string)
getup	POST	**x, isAjaxRequest** - see the following table showing common parameters.
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.

Table 6-79. Actions (continued)

Action	HTTP Request	Parameters
getgadgetprefs	POST	**x, isAjaxRequest** - see the following table showing common parameters.
notifications	POST	**x, isAjaxRequest** – see the following table showing common parameters. **subscribe** – **JSON** string containing subscribe options **unsubscribe** – when specified, designates an unsubscribe action **updateToken** – the update notification token (string) **adapter** – the name of the notification adapter (string) **eventSource** – the name of the notification event source (string) **alias** – notification subscription alias (string) **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 6-79. 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 6-80. 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}/{Applica
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **Application ID**, and **Application Environment**.

Table 6-81. 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}/{
```

Request elements

See *Application API Request Elements* for details about the following request elements: **Protocol**, **Worklight Server**, **ApplicationID**, and **Application Environment**.

Table 6-82. 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

Troubleshooting MobileFirst 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 11-37.

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 a MobileFirst Server upgrade with Derby as the database

If IBM MobileFirst Platform 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 IBM MobileFirst Platform Server installation.

About this task

During an upgrade of MobileFirst 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 MobileFirst Server and Application Center, stop the application server that runs the Application Center application.

WebSphere Application Server Liberty profile: troubleshooting Administration Services

In IBM MobileFirst Platform Foundation for iOS on Liberty profile, learn how to troubleshoot version incompatibility, JMX configuration, communication, and server farm topology.

If the MobileFirst runtime is not initialized correctly when you log in to MobileFirst Operations Console, the runtime is not displayed. You could receive the following error message:

```
The runtime environment has not initialized successfully; check server
logs.
```

When you start the Administration Services and MobileFirst runtimes, one or more exceptions are reported in the server logs of WebSphere Application Server Liberty profile, because of a configuration issue.

Check the server log file of WebSphere Application Server Liberty profile, by default, `messages.log`. In the case of a server farm topology, you must check the log file of each member of the farm.

Versions of Administration Services and runtimes

Check that the version of the Administration Services and the MobileFirst runtime are exactly the same.

Procedure

1. To check the versions of all the components, use the wladm **`show versions`** command. For more information about this command, see "Commands for troubleshooting" on page 12-60.

If the values of **`wladmVersion`** and **`productVersion`** of each project WAR file are not all the same, you must upgrade the component with the earlier version to the same version as the component with the later version.

2. Upgrade the earlier version. To upgrade the Administration Services or the runtimes, see "Upgrading the MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks" on page 7-46.

Troubleshooting the JMX configuration

You can encounter several exceptions in the Liberty profile server logs that are related to the JMX configuration.

Table 6-83. Errors in JMX configuration for WebSphere Application Server Liberty profile.

Message title	Error	Cause	Resolution
Invalid administrator user	`Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: CWWKX0215E: There was a problem with the user name or password provided. The server responded with code 401 and message 'Unauthorized'`	The value of the **`ibm.worklight.admin.jmx.user`** JNDI property is not an administrative Liberty profile user.	Edit the `server.xml` file and make sure that the user referenced in **`ibm.worklight.admin.jmx.user`** is defined in the **`<administrator-role>`** element.
SSL socket factory not found	`Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: java.lang.ClassNotFoundException: Cannot find the specified class com.ibm.websphere.ssl.protocol.SSLSocketFactory`	The IBM JDK cannot be used with the SSL socket factories of WebSphere Application Server Liberty profile.	For information about resolving this issue, see "Configuring Liberty profile when IBM JDK is used" on page 6-287.
No JMX connector configured	`Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: No JMX connector is configured`	The host name or the port number that is required for the JMX connection is not configured.	Edit the `server.xml` file and make sure that both the **`ibm.worklight.admin.jmx.port`** and the **`ibm.worklight.admin.jmx.host`** JNDI properties are defined.

Table 6-83. Errors in JMX configuration for WebSphere Application Server Liberty profile (continued).

Message title	Error	Cause	Resolution
Read timed out	`Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: Read timed out`	The JMX connection times out before the operation completes. By default, the JMX connection times out after one minute.	Edit the Liberty profile `jvm.options` file and add the following property: `-Dcom.ibm.ws.jmx.connector.client.res` The default value is 60000. Use a greater value. The following example uses three minutes. `-Dcom.ibm.ws.jmx.connector.client.res`
Invalid certification path	`Failed to obtain JMX connection to access an MBean. There might be a JMX configuration error: com.ibm.jsse2.util.h: PKIX path building failed: java.security.cert.CertPathBuilderException: unable to find valid certification path to requested target`	The SSL configuration of the Liberty profile server is not correct.	For instructions about how to resolve this issue, see Configuring secure JMX connection to the Liberty profile.
Connection exception	`java.net.ConnectException: Connection refused: connect`	The JMX connection fails.	Edit the `server.xml` file. Make sure that both the **`ibm.worklight.admin.jmx.port`** and the **`ibm.worklight.admin.jmx.host`** JNDI properties reference the local host, and that the https port number is defined in the <httpEndpoint> element.

Troubleshooting communication between the runtime and the Administration Services

Determine the causes of communication failure that lead to an exception reported in the server log.

At startup, each MobileFirst runtime must communicate with the Administration Services in JMX and HTTP or HTTPS for the purpose of synchronization. If communication fails, the runtime cannot initialize and the following exception is reported in the server log:

```
java.lang.RuntimeException: Timeout while waiting for the management
service to start up.120 secs.
```

This communication failure can be due to one of several causes. You can determine the cause from the messages in the server log that are reported before the exception.

Administration Services Mbean not found

If the log contains one of the following exceptions, the runtime has not been able to locate the Administration Services Mbean by name. Therefore, the Administration Services Mbean is not registered or is registered under a different name.

```
Caused by: javax.management.InstanceNotFoundException: CWWKX0217E: No MBean
is currently registered with the given ObjectName <Administration Services
Mbean name>
```

or an exception beginning:

```
javax.management.InstanceNotFoundException:
WebSphere:type=WorklightAdmin[_<environmentid>]
```

The registration of the Administration Services is logged in this way:

```
FWLSE2008I: MBean registration succeeded for:
com.worklight.common.server.jmx.api:type=WorklightAdmin[_<environmentid>],qualifier=
```

You can find the root cause by comparing this MBean name with the one referenced in the CWWKX0217E error and then checking in the Liberty Profile `server.xml` file the value of the JNDI properties **`ibm.worklight.admin.environmentid`** or, for a server farm topology, **`ibm.worklight.admin.serverid`**. These properties must have the same values for the Administration Services and the runtime. See “Stand-alone server topology” on page 6-9 and “Server farm topology” on page 6-11.

Administration Services endpoint not found

If the log contains the following exception, the endpoint of the Administration Services could not be determined.

```
Could not determine the own endpoint from the application server's
configuration.
```

By default, the Administration Services detects its own endpoint. If for some reason it cannot, or you want all HTTP requests to go through a web server, you must configure the endpoint properties in accordance with the information given in “Configuring the endpoint (Liberty profile)” on page 6-94.

Administration Services endpoint not accessible

If the log contains many information messages like `Accessing the ping service <ping endpoint>`, and does not contain an information message like `The ping service <ping endpoint> is available`, the Administration Services endpoint identified in the first kind of information message cannot be accessed.

Check that this endpoint is correct and whether it can be accessed from the server of WebSphere Application Server Liberty profile.

- If this endpoint references a web server and a firewall is used on this server, you must check that the firewall authorizes access from the server of WebSphere Application Server Liberty profile.
- If the protocol used for invoking the web server is SSL, you must check that SSL is correctly configured between the server of WebSphere Application Server Liberty profile and the web server.

See Configuring a web server plug-in for the Liberty profile.

Troubleshooting server farm topology

Learn how to troubleshoot when you get exceptions in a server farm.

When you have a server farm topology, exceptions are raised when the Administration Services MBean is registered and the farm configuration is not correct. For more information, see “Troubleshooting server farm configuration issues” on page 6-355.

WebSphere Application Server full profile: troubleshooting Administration Services in a stand-alone topology

Learn how to troubleshoot version incompatibility, JMX configuration, communication, and server farm topology in IBM MobileFirst Platform Foundation for iOS on WebSphere Application Server full profile, stand-alone server.

If the MobileFirst runtime is not initialized correctly when you log in to MobileFirst Operations Console, the runtime is not displayed. You could receive the following error message:

```
The runtime environment has not initialized successfully; check server
logs.
```

When you start the Administration Services and MobileFirst runtimes, one or more exceptions are reported in the server logs of WebSphere Application Server, because of a configuration issue.

Check the WebSphere Application Server server log file, by default, `SystemOut.log`. (In the case of a server farm topology, you must check the log file of each member of the farm.)

Versions of Administration Services and runtimes

Check that the version of the Administration Services and the MobileFirst runtime are exactly the same.

Procedure

1. To check the versions of all the components, use the wladm **`show versions`** command. For more information about this command, see “Commands for troubleshooting” on page 12-60.

If the values of **`wladmVersion`** and **`productVersion`** of each project WAR file are not all the same, you must upgrade the component with the earlier version to the same version as the component with the later version.

2. Upgrade the earlier version. To upgrade the Administration Services or the runtimes, see “Upgrading the MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks” on page 7-46.

Troubleshooting the JMX configuration

When the MobileFirst runtime starts, it must communicate by using JMX with the Administration Services instance that manages this runtime and that is deployed in the same JVM.

Table 6-84. Errors in JMX configuration for WebSphere Application Server full profile.

Message title	Error	Cause	Resolution
Invalid JMX Connector	`invalid JMX Connector <type>`	The value of the **`ibm.worklight.admin.jmx.connector`** JNDI property is invalid; it must be `SOAP` or `RMI`.	See "Stand-alone server topology" on page 6-9 for information about how to resolve this problem.
SOAP connection timeout	`SOAPException: faultCode=SOAP-ENV:Client; msg=Read timed out; targetException=java.net.SocketTimeoutException: Read timed out`	The JMX connection timed out before the operation completed. By default, the SOAP connection times out three minutes after the operation starts.	Increase the value of the **`com.ibm.SOAP.requestTimeout`** property in the WebSphere `soap.client.props` file. See SOAP connector properties.

Troubleshooting communication between the runtime and the Administration Services

Determine the causes of communication failure that lead to an exception reported in the server log.

At startup, each MobileFirst runtime must communicate with the Administration Services in JMX and HTTP or HTTPS for the purpose of synchronization. If communication fails, the runtime cannot initialize and the following exception is reported in the server log:

```
java.lang.RuntimeException: Timeout while waiting for the management
service to start up.120 secs.
```

This communication failure can be due to one of several causes. You can determine the cause from the messages in the server log that are reported before the exception.

Administration Services Mbean not found

If the log contains one of the following exceptions, the runtime has not been able to locate the Administration Services Mbean by name. Therefore, the Administration Services Mbean is not registered or is registered under a different name.

```
Caused by: javax.management.InstanceNotFoundException: CWWKX0217E: No MBean
is currently registered with the given ObjectName <Administration Services
Mbean name>
```

or an exception beginning:

```
javax.management.InstanceNotFoundException:
WebSphere:type=WorklightAdmin[_<environmentid>]
```

The registration of the Administration Services is logged in this way:

```
FWLSE2008I: MBean registration succeeded for:
com.worklight.common.server.jmx.api:type=WorklightAdmin[_<environmentid>],qualifier=
```

You can find the root cause by comparing this MBean name with the one referenced in the CWWKX0217E error and then checking in the WebSphere Application Server administration console the value of the JNDI property **ibm.worklight.admin.environmentid** (and for a server farm topology, **ibm.worklight.admin.serverid**). These properties must have the same values for the Administration Services and the runtime. See "Stand-alone server topology" on page 6-9 and "Server farm topology" on page 6-11.

Administration Services endpoint not found

If the log contains the following exception, the endpoint of the Administration Services could not be determined.

```
Could not determine the own endpoint from the application server's
configuration.
```

By default, the Administration Services detects its own endpoint by using the WebSphere Application Server virtual host definition mapped to the Administration Services application.

You must first check that the virtual host configuration, by default, set as default_host, contains at least one virtual host alias with the HTTP or HTTPS port number of the WebSphere Application Server server. If no entry is defined, you must create one as described in Configuring virtual hosts.

If for some reason detection fails, or you want all HTTP requests to go through a web server, you must configure the endpoint properties in accordance with the information given in "Configuring the endpoint (WebSphere Application Server full profile)" on page 6-93.

Administration Services endpoint not accessible

If the log contains many information messages like `Accessing the ping service <`*`ping endpoint`*`>`, and does not contain an information message like `The ping service <`*`ping endpoint`*`> is available`, the Administration Services endpoint identified in the first kind of information message cannot be accessed.

Check that this endpoint is correct and whether it can be accessed from the server of WebSphere Application Server.

- If this endpoint references a web server and a firewall is used on this server, you must check that the firewall authorizes access from the server of WebSphere Application Server.
- If the protocol used for invoking the web server is SSL, you must check that SSL is correctly configured between the server of WebSphere Application Server and the web server. See Configuring the web server plug-in for Secure Sockets Layer.

Troubleshooting server farm topology

Learn how to troubleshoot when you get exceptions in a server farm.

When you have a server farm topology, exceptions are raised when the Administration Services MBean is registered and the farm configuration is not correct. For more information, see "Troubleshooting server farm configuration issues" on page 6-355.

WebSphere Application Server Network Deployment: troubleshooting Administration Services

Learn how to troubleshoot version incompatibility, JMX configuration, and communication in IBM MobileFirst Platform Foundation for iOS on WebSphere Application Server Network Deployment.

If the MobileFirst runtime is not initialized correctly when you log in to MobileFirst Operations Console, the runtime is not displayed. You could receive the following error message:

```
The runtime environment has not initialized successfully; check server
logs.
```

When you start the Administration Services and MobileFirst runtimes, one or more exceptions are reported in the server logs of WebSphere Application Server Network Deployment, because of a configuration issue.

Check the WebSphere Application Server Network Deployment server log file, by default, `SystemOut.log`. In the case of a clustered topology, you must check the log file of each cluster member. In an asymmetric deployment, the Administration Services are not deployed in the same JVM as the runtimes; you must check all the logs where these components are deployed.

Versions of Administration Services and runtimes

Check that the version of the Administration Services and the MobileFirst runtime are exactly the same.

Procedure

1. To check the versions of all the components, use the wladm **`show versions`** command. For more information about this command, see "Commands for troubleshooting" on page 12-60.

If the values of **`wladmVersion`** and **`productVersion`** of each project WAR file are not all the same, you must upgrade the component with the earlier version to the same version as the component with the later version.

2. Upgrade the earlier version. To upgrade the Administration Services or the runtimes, see "Upgrading the MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks" on page 7-46.

Troubleshooting the JMX configuration

When the MobileFirst runtime starts, it must communicate by using JMX with the Administration Services instance that manages this runtime.

Table 6-85. Errors in JMX configuration for WebSphere Application Server Network Deployment

Message title	Error	Cause	Resolution
Invalid JMX Connector	`invalid JMX Connector <type>`	The value of the **`ibm.worklight.admin.jmx.connector`** JNDI property is invalid; it must be `SOAP` or `RMI`.	See “WebSphere Application Server Network Deployment topologies” on page 6-15 for information about how to resolve this problem.
SOAP connection failed	`com.ibm.websphere.management.exception.ConnectorException: ADMC0016E: The system cannot create a SOAP connector to connect to host <host> at port <port>`	The value of the **`ibm.worklight.admin.jmx.host`** or the **`ibm.worklight.admin.jmx.port`** JNDI property is invalid.	See “WebSphere Application Server Network Deployment topologies” on page 6-15 for information about how to resolve this problem.
SOAP connection timeout	`SOAPException: faultCode=SOAP-ENV:Client; msg=Read timed out; targetException=java.net.SocketTimeoutException: Read timed out`	The JMX connection timed out before the operation completed. By default, the SOAP connector times out three minutes after the operation starts.	Increase the value of the **`com.ibm.SOAP.requestTimeout`** property in the WebSphere `soap.client.props` file. See SOAP connector properties.

Troubleshooting communication between the runtime and the Administration Services

Determine the causes of communication failure that lead to an exception reported in the server log.

At startup, each MobileFirst runtime must communicate with the Administration Services in JMX and HTTP or HTTPS for the purpose of synchronization. If communication fails, the runtime cannot initialize and the following exception is reported in the server log:

```
java.lang.RuntimeException: Timeout while waiting for the management
service to start up.120 secs.
```

This communication failure can be due to one of several causes. You can determine the cause from the messages in the server log that are reported before the exception.

Administration Services Mbean not found

If the log contains one of the following exceptions, the runtime has not been able to locate the Administration Services Mbean by name. Therefore, the Administration Services Mbean is not registered or is registered under a different name.

```
Caused by: javax.management.InstanceNotFoundException: CWWKX0217E: No MBean
is currently registered with the given ObjectName <Administration Services
Mbean name>
```

or an exception beginning:

```
javax.management.InstanceNotFoundException:
WebSphere:type=WorklightAdmin[_<environmentid>]
```

The registration of the Administration Services is logged in this way:

```
FWLSE2008I: MBean registration succeeded for:
com.worklight.common.server.jmx.api:type=WorklightAdmin[_<environmentid>],qualifier=
```

You can find the root cause by comparing this MBean name with the one referenced in the CWWKX0217E error and then checking in the WebSphere Application Server administration console the value of the JNDI property **ibm.worklight.admin.environmentid**. These properties must have the same values for the Administration Services and the runtime. See "WebSphere Application Server Network Deployment topologies" on page 6-15.

Administration Services endpoint not found

If the log contains the following exception, the endpoint of the Administration Services could not be determined.

```
Could not determine the own endpoint from the application server's
configuration.
```

By default, the Administration Services detects its own endpoint by using the WebSphere Application Server virtual host definition mapped to the Administration Services application.

You must first check that the virtual host configuration, by default, set as default_host, contains at least one virtual host alias with the HTTP or HTTPS port number of the WebSphere Application Server server. In the case of a clustered topology, you must check the virtual host configuration for every cluster member. If no entry is defined, you must create one as described in Configuring virtual hosts.

If for some reason detection fails and you do not want all HTTP requests to go through a web server, you must configure the endpoint properties at the JVM level. In the case of a clustered environment, you must configure the endpoint properties for the JVM of every cluster member.

1. Log in to the WebSphere Application Server administration console.
2. Select **Servers** > **Server Types** > **WebShere application servers** > *server_name*. .
3. Under **Server Infrastructure**, select **Java and process management** > **Process definition** > **Java virtual machine** > **Custom properties**.
4. Add the following properties:

Property name	Value
ibm.worklight.admin.proxy.protocol	http
ibm.worklight.admin.proxy.host	hostname of the server
ibm.worklight.admin.proxy.port	http port number of the server

5. Save the configuration and restart the server.

In the case of a clustered environment, you must perform these steps for all the cluster members where the Administration Services application is deployed.

If you want all the HTTP requests to go through a web server, you must configure the endpoint properties in accordance with the information given in "Configuring the endpoint (WebSphere Application Server full profile)" on page 6-93.

Administration Services endpoint not accessible

If the log contains many information messages like `Accessing the ping service` *<ping endpoint>*, and does not contain an information message like `The ping service` *<ping endpoint>* `is available`, the Administration Services endpoint identified in the first kind of information message cannot be accessed.

Check that this endpoint is correct and whether it can be accessed from the server in WebSphere Application Server Network Deployment.

- If this endpoint references a web server and a firewall is used on this server, you must check that the firewall authorizes access from the server in WebSphere Application Server Network Deployment.
- If the protocol used for invoking the web server is SSL, you must check that SSL is correctly configured between the server in WebSphere Application Server Network Deployment and the web server. See Configuring the web server plug-in for Secure Sockets Layer.

Apache Tomcat: troubleshooting Administration Services

In IBM MobileFirst Platform Foundation for iOS on Apache Tomcat, learn how to troubleshoot version incompatibility, JMX configuration, communication, and server farm topology.

If the MobileFirst runtime is not initialized correctly when you log in to MobileFirst Operations Console, the runtime is not displayed. You could receive the following error message:

```
The runtime environment has not initialized successfully; check server
logs.
```

When you start the Administration Services and MobileFirst runtimes, one or more exceptions are reported in the Apache Tomcat server logs, because of a configuration issue.

Check the Apache Tomcat server log file, by default, `catalina.log`. In the case of a server farm topology, you must check the log file of each member of the farm.

Versions of Administration Services and runtimes

Check that the version of the Administration Services and the MobileFirst runtime are exactly the same.

Procedure

1. To check the versions of all the components, use the wladm **`show versions`** command. For more information about this command, see "Commands for troubleshooting" on page 12-60.

If the values of **`wladmVersion`** and **`productVersion`** of each project WAR file are not all the same, you must upgrade the component with the earlier version to the same version as the component with the later version.

2. Upgrade the earlier version. To upgrade the Administration Services or the runtimes, see "Upgrading the MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks" on page 7-46.

Troubleshooting the JMX configuration

You can encounter several exceptions in the Apache Tomcat server logs that are related to the JMX configuration.

When the MobileFirst runtime starts, it uses JMX to communicate with the Administration Services deployed in the same JVM. Exceptions are reported in the `catalina.log` and the `output.log` files.

Table 6-86. Errors in JMX configuration on Apache Tomcat

Message title	Error	Cause	Resolution
RMI server hostname not configured	`The JVM is not properly configured to support JMX over RMI. The JVM property java.rmi.server.hostname must be defined`	The value of the **`java.rmi.server.hostname`** System property is not set.	For information about resolving this issue, see "Configuring Apache Tomcat" on page 6-54
RMI port number not configured	`The JVM is not properly configured to support JMX over RMI. If the Tomcat instance is not running behind a firewall, the JVM property com.sun.management.jmxremote.port must be defined`	The value of the **`com.sun.management.jmxremote.port`** System property is not set.	For information about resolving this issue, see "Configuring Apache Tomcat" on page 6-54.
Unknown host exception	`java.rmi.UnknownHostException: Unknown host` *`<hostname>`*	The hostname referenced by the **`java.rmi.server.hostname`** System property is not accessible.	The value of the **`java.rmi.server.hostname`** System property must be the hostname of the Apache Tomcat server.
Port in use	`java.rmi.server.ExportException: Port already in use (emitted in the outputfile.log)`	Another process is already using the port referenced by the **`com.sun.management.jmxremote.port`** System property.	Set the value of the **`com.sun.management.jmxremote.port`** System property to a port number that is not already in use.
Connection exception	`java.io.IOException: Failed to retrieve RMIServer stub java.net.ConnectException: Connection refused: connect`	The value of the `ibm.worklight.admin.rmi.registryPort` JNDI property refers to an invalid port.	For information about resolving this issue, see "Configuring Apache Tomcat" on page 6-54.

Table 6-86. Errors in JMX configuration on Apache Tomcat (continued)

Message title	Error	Cause	Resolution
Socket timeout	`java.net.SocketTimeoutException: Read timed out`	The JMX connection times out before the operation completes. By default, the RMI connection never times out.	The **`sun.rmi.transport.tcp.responseTimeout`** JVM property must be set to `0`, which means that no timeout is set, or its value must be increased.
SSL handshake Exception	`javax.net.ssl.SSLHandshakeException: sun.security.validator.ValidatorException: PKIX path building failed`	The truststore configuration is not correct. Possibly, the Apache Tomcat certificate is not present in the trust store referenced by the **`javax.net.ssl.trustStore`** System property	For information about resolving this issue, see "Configuring Apache Tomcat" on page 6-54.

Troubleshooting communication between the runtime and the Administration Services

Determine the causes of communication failure that lead to an exception reported in the server log.

At startup, each MobileFirst runtime must communicate with the Administration Services in JMX and HTTP or HTTPS for the purpose of synchronization. If communication fails, the runtime cannot initialize and the following exception is reported in the server log:

```
java.lang.RuntimeException: Timeout while waiting for the management
service to start up.120 secs.
```

This communication failure can be due to one of several causes. You can determine the cause from the messages in the server log that are reported before the exception.

Administration Services Mbean not found

If the log contains one of the following exceptions, the runtime has not been able to locate the Administration Services Mbean by name. Therefore, the Administration Services Mbean is not registered or is registered under a different name.

```
Caused by: javax.management.InstanceNotFoundException: CWWKX0217E: No MBean
is currently registered with the given ObjectName <Administration Services
Mbean name>
```

or an exception beginning:

```
com.worklight.common.server.jmx.api:type=WorklightAdmin[_<environmentid>]
```

The registration of the Administration Services is logged in this way:

```
FWLSE2008I: MBean registration succeeded for:
com.worklight.common.server.jmx.api:type=WorklightAdmin[_<environmentid>],qualifier=
```

You can find the root cause by comparing this MBean name with the one referenced in the CWWKX0217E error and then checking in the Apache Tomcat `server.xml` file the value of the JNDI properties **`ibm.worklight.admin.environmentid`** or, for a server farm topology, **`ibm.worklight.admin.serverid`**. These properties must have the same values for the Administration Services and the runtime. See "Stand-alone server topology" on page 6-9 and "Server farm topology" on page 6-11.

Administration Services endpoint not found

If the log contains the following exception, the endpoint of the Administration Services could not be determined.

```
Could not determine the own endpoint from the application server's
configuration.
```

By default, the Administration Services detects its own endpoint. If for some reason it cannot, or you want all HTTP requests to go through a web server, you must configure the endpoint properties in accordance with the information given in "Configuring the endpoint (Apache Tomcat)" on page 6-95.

Administration Services endpoint not accessible

If the log contains many information messages like `Accessing the ping service <`*`ping endpoint`*`>`, and does not contain an information message like `The ping service <`*`ping endpoint`*`> is available`, the Administration Services endpoint identified in the first kind of information message cannot be accessed.

Check that this endpoint is correct and whether it can be accessed from the Apache Tomcat server.

- If this endpoint references a web server and a firewall is used on this server, you must check that the firewall authorizes access from the Apache Tomcat server.
- If the protocol used for invoking the web server is SSL, you must check that SSL is correctly configured between the Apache Tomcat server and the web server.

Troubleshooting server farm topology

Learn how to troubleshoot when you get exceptions in a server farm.

When you have a server farm topology, exceptions are raised when the Administration Services MBean is registered and the farm configuration is not correct. For more information, see "Troubleshooting server farm configuration issues."

Troubleshooting server farm configuration issues

When you start the Administration Services and the MobileFirst runtime environments, several exception types can be emitted in the application server logs if the configuration of the server farm is incorrect.

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. Restart the application server. For more information, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

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 situation occurs 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 as one defined in another server of the farm.

You must configure the property **`ibm.worklight.admin.serverid`** in the application server. This property must be unique among the servers of the farm. Restart the application server.

Use of a Derby database

MBeanRegistrationException "Derby databases are not supported in a farm of servers".

You must use a DB2, Oracle, or MySQL database for the runtime and Administration Services databases, because Derby is not supported in server farm topology.

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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS is uninstalled, the basic registry that was created during the installation by the IBM MobileFirst Platform 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 IBM MobileFirst Platform Operations 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 "appcenter
  <user name="appcenteradmin" password="admin"/>
  <user name="demo" password="demo"/>
  <group name="appcentergroup">
    <member name="appcenteradmin"/>
    <member name="demo"/>
  </group>
</basicRegistry>
```

2. When you uninstall IBM MobileFirst Platform Foundation for iOS, the uninstaller of MobileFirst 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.

Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0

This section contains the procedures for upgrading from IBM Worklight V5.0.6 or later to V7.1.0 and migrating the applications you created in earlier versions of the product to work with IBM MobileFirst Platform Foundation for iOS V7.1.0.

About this task

Upgrading from one version of the product 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 to V7.1.0 of MobileFirst Server, and how to migrate your applications for V7.1.0.

Version compatibility

Compatibility among different versions of the IBM MobileFirst Platform Foundation for iOS 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 7-1. Version compatibility rules

Description	Compatibility rule	Examples
The server and client have the 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.	7.1.0.0 server is compatible with 7.1.0.0 client.
The server is newer than the client.	Compatible.	7.1.0.0 server is compatible with 7.0.0.0 client
The server is older than the client.	Not compatible.	7.0.0.0 server is not compatible with 7.1.0.0 client

Table 7-1. Version compatibility rules (continued)

Description	Compatibility rule	Examples
The server artifacts were created with older version of MobileFirst Studio or MobileFirst Platform Command Line Interface for iOS 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 earlier than 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 later 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 later 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 • Later server version that the `.war` file is migrated to • Any version earlier than the initial version The artifacts 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 MobileFirst Server" on page 11-37.	Example 1: Artifacts built with MobileFirst Studio 7.0.0.0 can run on server 7.1.0.0. However, the artifacts will behave as though they are running on server version 7.0.0.0. Example 2: Initially, a version 6.2.0.1 `.war` file can run the artifacts from 6.2.01 and earlier versions on a 6.2.0.1 server. If this `.war` file is migrated to 7.1, then the migrated `.war` file can run 7.1 artifacts. However, this `.war` file cannot run MobileFirst Server 7.0.0.0 artifacts.
Server artifacts created with a later version of MobileFirst Studio or the MobileFirst Platform Command Line Interface for iOS than the version of the server.	Not compatible.	Artifacts built with MobileFirst Studio 7.1.0.0 cannot run on server 7.0.0.0.

Table 7-1. Version compatibility rules (continued)

Description	Compatibility rule	Examples
Direct Update feature	If the version of MobileFirst Studio or the MobileFirst Platform Command Line Interface for iOS that was used to build an update package differs from the version of MobileFirst Studio or the MobileFirst Platform Command Line Interface for iOS that was used to build the original application package then the update will not be applied.	Original application was built with MobileFirst Studio 7.0.0.0; update was built with MobileFirst Studio 7.1.0.0. 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. Version numbers Before 6.3 apply to Worklight products. Version numbers of 6.3 and later apply to MobileFirst products.

Table 7-2. 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 following artifact versions (application and adapter files created with this version of Studio or command line)
7.1.0	7.1.0	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0, 7.0.0, 7.1.0
7.1.0	7.0.0 migrated to 7.1.0	.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0, 7.0.0
7.1.0	6.3.0 migrated to 7.1.0	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0
7.1.0	6.2.0 migrated to 7.1.0	6.2.0
7.1.0	6.10 migrated to 7.1.0	6.1.0
7.1.0	6.0.0 migrated to 7.1.0	6.0.0
7.1.0	5.0.6 migrated to 7.1.0	5.0.6
7.0.0	7.0.0	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0, 7.0.0
7.0.0	6.3.0 migrated to 7.0.0	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0
7.0.0	6.2.0 migrated to 7.0.0	6.2.0
7.0.0	6.10 migrated to 7.0.0	6.1.0
7.0.0	6.0.0 migrated to 7.0.0	6.0.0
7.0.0	5.0.6 migrated to 7.0.0	5.0.6
6.3.0	6.3.0	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.3.0
6.3.0	6.2.0 migrated to 6.3.0	6.2.0
6.3.0	6.1.0 migrated to 6.3.0	6.1.0
6.3.0	6.0.0 migrated to 6.3.0	6.0.0
6.3.0	5.0.6 migrated to 6.3.0	5.0.6
6.2.0.1	6.2.0.1	5.0.6, 6.0.0, 6.1.0, 6.2.0, 6.2.0.1
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

Table 7-2. Server, project and artifact compatibility (continued)

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 following artifact versions (application and adapter files created with this version of Studio or command line)
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

Migrating projects to IBM MobileFirst Platform Foundation for iOS V7.1.0

How to migrate your existing projects to IBM MobileFirst Platform Foundation for iOSV7.1.0

Migrating projects to V7.1.0 using MobileFirst Platform Command Line Interface for iOS

If you are using IBM MobileFirst Platform Command Line Interface for iOS to develop an IBM MobileFirst Platform Foundation for iOS project that is from any release before V7.1.0, your project is automatically migrated to V7.1.0

When you run the `mfp add` command or the `mfp build` command, or when the MobileFirst Server starts, your MobileFirst project is migrated to V7.1.0.

Migrating MobileFirst Native API projects

After you migrate your existing MobileFirst native API project, you must replace SDK and configuration files with later versions. For iOS Native API projects see "Upgrading existing native iOS applications" on page 7-5.

iOS: ARC-enablement for upgraded iOS projects

The MobileFirst upgrade process does not enable ARC (Automatic Reference Counting) by default in existing iOS projects. Although ARC is enabled for the MobileFirst 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. Starting with 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 V7.1.0 iOS environments and the iOS environments that are migrated from previous versions of IBM MobileFirst Platform Foundation for iOS have the `-force_load` flag specified for the following libraries:

- `libCordova.a`
- `libTLFLib.a` (added by the Tealeaf® SDK optional feature)
- `libfipshttp.a` (added by the FIPS 140-2 optional feature)

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). 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
   ```

2. If Xcode was previously run on the generated `.xcodeproj` project, then you can specify the `scheme`:

   ```
   xcodebuild -project <PathToProject>/TestProjectTestAppIphone.xcodeproj -scheme TestProjectTest
   ```

iOS: 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.

     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.

Upgrading existing native iOS applications

To work with an existing native iOS project that was created with IBM MobileFirst Platform Foundation for iOS V7.0.0 or earlier to V7.1.0 (or more recent interim fixes), you must modify the project by manually modifying your project in Xcode, or by using CocoaPods.

The latest interim fixes introduced a number of changes for adapting to the latest version of iOS (iOS 9) and Xcode 7.1 and 7.1.1. CocoaPods also recently released a new version that changes the way the pods are installed.

In order to correctly configure Xcode 7.1 and 7.1.1 to work with the latest interim fixes, you need to understand what settings Xcode uses and why some settings can be problematic.

The latest interim fixes deliver the necessary libraries in the form of frameworks. Frameworks can contain a number of resources. Frameworks contain library as well as header files. One advantage of this method of delivering the SDK is that multiple resources are organized together, reducing the amount of linking required within the Xcode project.

It is important to clean up all unnecessary libraries, header files, and their links within Xcode before installing the latest interim fix. Xcode 9 continues to work with libraries and header files, so some non-IBM MobileFirst Platform Foundation for iOS library or header files may need to remain.

Upgrading existing native iOS applications to IBM MobileFirst Platform Foundation for iOS version V7.1.0 manually

You can upgrade your existing native iOS application manually, in Xcode, so that you can work with it in IBM MobileFirst Platform Foundation for iOS V7.1.0.

Before you begin

- You must have an existing native iOS project that was created with IBM MobileFirst Platform Foundation for iOS 7.0.0 or earlier.
- You must have already created a new, V7.1.0 or later MobileFirst native API project with the MobileFirst Platform Command Line Interface for iOS so that you can copy certain required files to your Xcode project. For more information, see the "`create`" on page 15-6 command.

Procedure

1. Manually delete the `WorklightAPI` folder under the main Xcode project folder.
2. Replace the `Worklight.plist` file in your Xcode project folder with the `Worklight.plist` file from your new 7.1 MobileFirst Studio project.
3. In the Xcode **Build** Phases tab, **Link Binary With Libraries** section, remove all references, including the `libWorklightStaticLibProjectNative.a` reference.
4. Add the IBM MobileFirst Platform Foundation for iOS 7.1 SDK.
   a. Find the `WorklightAPI` folder in your new IBM MobileFirst Platform Foundation for iOS 7.1 or MobileFirst Studio project folder and manually copy it to your main Xcode project folder.
   b. Add the reference to your project: Choose **Add Files** from the **File** menu and navigate to the Worklight folder in your Xcode project folder.
5. In the **Search Paths** section of the **Build Settings** tab:
   a. Check that the **Header Search Paths** values have been removed.
   b. Verify that the **Framework Search Paths** value is `$(PROJECT_DIR)/WorklightAPI/Frameworks`.
   c. The **Library Search Paths** values can also be removed.
6. If you are using Swift, make sure the `$SRCROOT/WorklightAPI/include/WLSwiftBridgingHeader.h` has been removed from **Swift compiler** section in the **Build Settings** tab (**Objective-C Bridging Header**). This header is included in the newly added IBM MobileFirst Platform Foundation for iOS 7.1 frameworks in the **Build Phases** tab, **Link Binary With Libraries** section
7. Link the required frameworks, and possibly libraries, and headers in the **Build Phases** tab, **Link Binary With Libraries** section:

a. Make sure the IBM MobileFirst Platform Foundation for iOS frameworks have been linked:
   - `IBMMobileFirstPlatformFoundation.framework`

   Important: If you are using the latest interim fix link the following frameworks:
   - `openssl.framework`
   - If you want to enable the JSONStore feature (which is optional in the latest interim fix) link these additional files:
     - `IBMMobileFirstPlatformFoundationJSONStore`
     - `SQLCipher.framework`

   If the values are set correctly in the **Framework Search Paths** (see above), the above values should appear in the **Link Binary With Libraries**.

b. Add additional required iOS frameworks, `dylib` files, and libraries (some libraries will already be linked):
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `SystemConfiguration.framework`
   - `libstdc++.dylib`
   - `libz.dylib`
   - `libc++.dylib`

   Note: From Xcode 7, the `*.tbd` replace the `*.dylib` files.

8. In the **Linking** section of the **Build Settings** tab make sure the **Other Linker Flags** value is set to `-ObjC`.
9. In your code import the new MobileFirst framework
   - If you are using Objective C:
     ```
     #import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
     ```
   - If you are using Swift:
     ```
     import IBMMobileFirstPlatformFoundation
     ```
10. If you are using IBM MobileFirst Platform Foundation for iOS and are using the JSONStore feature, replace all of the existing JSONStore imports of headers with a single entry of the following new umbrella header:

    For Objective C
    ```
    #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.
    ```
    For Swift
    ```
    import IBMMobileFirstPlatformFoundationJSONStore
    ```
11. Beginning with Xcode 7 TLS must be enforced, see “Enforcing TLS-secure connections in iOS apps” on page 8-19.
12. From Xcode 7 bitcode is set as a build default and must be disabled. For more information see “Disabling bitcode in Xcode builds” on page 8-20.

Results

Your application is now upgraded so that it will work with the IBM MobileFirst Platform Foundation, V7.1.0 iOS SDK.

Upgrading existing native iOS applications to IBM MobileFirst Platform Foundation for iOS V7.1.0 with CocoaPods

You can upgrade your existing native iOS application IBM MobileFirst Platform Foundation for iOS to work with V7.1.0 by making some changes in Xcode project, adding the IBM MobileFirst Platform Foundation iOS SDK with CocoaPods and adjusting your code.

Before you begin

- You must have CocoaPods installed in your development environment. For more information, see the "Getting Started" guide for CocoaPods installation.

About this task

Procedure

1. Make sure that your app is still registered on the server according to the `worklight.plist` client property file in your Xcode project. For example, as part of your upgrade, if you are using a new MobileFirst Studio or MobileFirst Platform Command Line Interface for iOS project, make sure you are using the version of the `worklight.plist` that corresponds to the server registration.
2. Open your project in Xcode.
3. If MobileFirst SDK was previously installed manually (not with CocoaPods), delete the `WorklightAPI` folder (move it to Trash).
4. Modify the **Build Settings** setup:
   a. If it exists, remove `$(SRCROOT)/WorklightAPI/include` from the **Header Search Path**.
   b. If it exists, remove `$(PROJECTDIR)/WorklightAPI/frameworks` from the **Frameworks Search Path**.
5. If you are using Swift, make sure the `$SRCROOT/WorklightAPI/include/WLSwiftBridgingHeader.h` has been removed from **Swift compiler** section from the **Build Settings** tab (**Objective-C Bridging Header**). The header will be installed as part of the new frameworks.
6. In the **Build Phases** tab, remove any existing MobileFirst links. These are the libraries that may have been previously installed for MobileFirst.
   - `libWorklightStaticLibProjectNative.a`
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `sqlcipher.framework`
   - `libstdc++.6.dylib`
   - `libz.dylib`
7. Close Xcode.
8. Get the IBM MobileFirst Platform Foundation iOS SDK from CocoaPods. To get the SDK, complete the following steps:
   a. Open **Terminal** at the location of your new Xcode project.
   b. Run the **`pod init`** command to create a `podfile` file.
   c. Open the `Podfile` file that is in the root of the project with a text editor.
   d. Comment out or remove the existing content.

e. Add the following lines, including the iOS version, and save the changes:

```
use_frameworks!
platform :ios, 9.0
pod 'IBMMobileFirstPlatformFoundation'
```

Note: The pod examples assume you are upgrading to the latest version of the MobileFirst. If you are are not using the latest version of MobileFirst, you need to indicate the version. For example, for importing the latest pod for 7.1.0 `IBMMobileFirstPlatformFoundation` the line would look like this:

```
pod 'IBMMobileFirstPlatformFoundation', '~> 7.1.0'
```

f. Specify additional pods in the file, for additional functionality required by your app. For example, if your app uses Cloudant for data storage, the podfile might look like:

```
source 'https://github.com/CocoaPods/Specs.git'
use_frameworks!
platform :ios, 9.0
pod 'IBMMobileFirstPlatformFoundation'
pod 'CloudantToolkitLocal'
```

If you are using the latest interim fix for IBM MobileFirst Platform Foundation for iOS and your app uses JSONStore, the podfile might look like:

```
source 'https://github.com/CocoaPods/Specs.git'
use_frameworks!
platform :ios, 9.0
pod 'IBMMobileFirstPlatformFoundation'
pod 'IBMMobileFirstPlatformFoundationJSONStore'
```

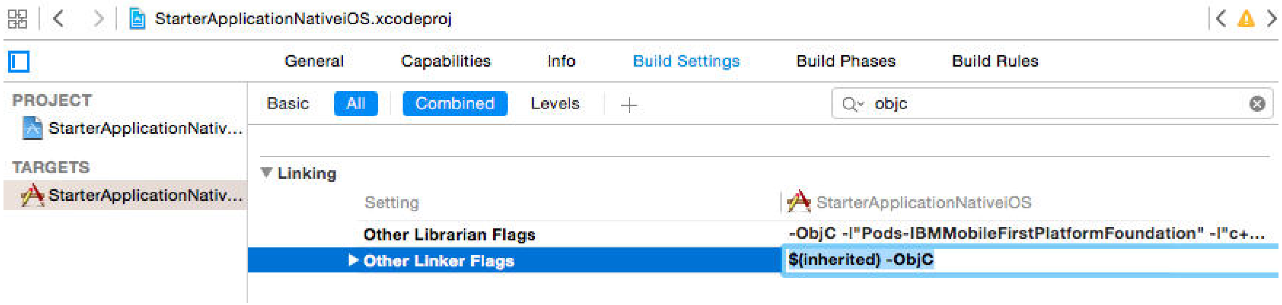
Note: If the podfile in this example specified only the `CloudantToolkitLocal` or `IBMMobileFirstPlatformFoundationJSONStore` pods and not the `IBMMobileFirstPlatformFoundation` pod, the results would be the same due to the built-in dependencies.

g. Verify that the Xcode project is closed.

h. Run the **`pod install`** command. (If IBM MobileFirst Platform Foundation, V7.1.0 was installed previously with CocoaPods run the **`pod update`** command). This command installs the MobileFirst SDK `IBMMobileFirstPlatformFoundation.framework` and any other frameworks that are specified in the podfile and their dependencies. It then generates the pods project, and integrates the client project with the MobileFirst SDK.

9. Open your `*ProjectName*.xcworkspace` file in Xcode by typing `open *ProjectName*.xcworkspace` from a command line. This file is in the same folder as the `*ProjectName*.xcodeproj` file.

10. In the **Build Settings** tab, under **Other Linker Flags**, add `$(inherited)` value to the -ObjC flag. For example:


StarterApplicationNativeiOS.xcodeproj
General
Capabilities
Info
Build Settings
Build Phases
Build Rules
Basic
All
Combined
Levels
objc
PROJECT
StarterApplicationNativ...
TARGETS
StarterApplicationNativ...
Linking
Setting
StarterApplicationNativeiOS

Setting	StarterApplicationNativeiOS
Other Librarian Flags	-ObjC -l"Pods-IBMMobileFirstPlatformFoundation" -l"c+...
Other Linker Flags	$(inherited) -ObjC

Figure 7-1. Adding $(inherited) to ObjC flag in Xcode Build Settings

11. In the **Search Paths** section of the **Build Settings** tab, make sure the **Header Search Paths** value has been cleared.
12. In your code, replace all of the existing MobileFirst imports of headers with frameworks.:

    For Objective C:

    ```
    #import <IBMMobileFirstPlatformFoundation/IBMMobileFirstPlatformFoundation.h>
    ```

    For Swift:

    ```
    import IBMMobileFirstPlatformFoundation
    ```

13. If you are using IBM MobileFirst Platform Foundation for iOS and are using the JSONStore feature, replace all of the existing JSONStore imports of headers with a single entry of the framework import:

    For Objective C

    ```
    #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.h>
    ```

    For Swift

    ```
    import IBMMobileFirstPlatformFoundationJSONStore
    ```

14. From Xcode 7 TLS needs to be enforced, see "Enforcing TLS-secure connections in iOS apps" on page 8-19.
15. From Xcode 7 bitcode is set as a build default and must be disabled. For more information see "Disabling bitcode in Xcode builds" on page 8-20.

Results

Your application is now upgraded so that it will work with the IBM MobileFirst Platform Foundation, V7.1.0 iOS SDK.

Upgrading projects to work in session-independent mode

Upgrading projects that were session-dependent in previous versions so that they work with IBM MobileFirst Platform Foundation for iOS V7.1.0 in session-independent mode.

About this task

Session-independent mode was introduced as the *default* mode for MobileFirst Server, starting with V7.1.0. For more information about session-independent mode, see "Session-independent mode" on page 8-108.

Note: Clients that were created using a version of IBM MobileFirst Platform Foundation for iOS earlier than V7.1.0 do not work with a MobileFirst Server that is operating in the new session-independent mode. So, to serve users who are using the session-independent mode, as well as existing users who are using session-dependent mode, you must deploy the new V7.1.0 WAR with its V7.1.0 artifacts, but **WITHOUT** removing the old WAR.

Tip: If you choose to move all your users to session-independent mode, for instance, to enable IT to scale down servers dynamically, you may consider remotely disabling the old apps in the old WAR.

Procedure

1. Import your old project into MobileFirst Studio V7.1.0. Alternatively, if you are using MobileFirst Platform Command Line Interface for iOS, make sure you have V7.1.0 installed, then run the `mfp push` command.
2. Inspect the settings for the session dependency properties in the `worklight.properties` file of your project.

   Ensure that the following parameters appear and are configured as shown:

   - `mfp.session.independent=true`.
   - `mfp.attrStore.type=database`. Other options are: `extremescale` and `datacache` (if you are using Bluemix). Make sure that `mfp.attrStore.type` is not set to `HttpSession`, which does not work in session-independent mode.

   For more information, see "Configuring session dependency" on page 11-54.
3. Inspect the expiration settings of the built-in realms in the `worklight.properties` file.

   Ensure that the following parameters appear. Either set a positive expiration value for each, or use the following default values:

   ```
   wl.realm.expiration.directUpdate=3600
   wl.realm.expiration.remoteDisable=300
   wl.realm.expiration.deviceAutoProvisioning=3600
   wl.realm.expiration.deviceNoProvisioning=3600
   wl.realm.expiration.antiXSRF=3600
   wl.realm.expiration.authenticity=3600
   wl.realm.expiration.anonymousUser=3600
   ```
4. Inspect the `loginModule` elements in the `authenticationConfig.xml` file of your project. Change the `expirationInSeconds` attribute value from `-1` to your desired realm expiration, or completely remove this attribute to use a default expiration of 3600 seconds. For more information, see "Configuring login modules" on page 8-263.
5. Ensure that the apps in your project conform to the requirements in the section "Developing apps to work in IBM MobileFirst Platform Foundation V7.1.0" under Session-independent mode.

Migrating IBM SmartCloud Analytics Embedded to IBM MobileFirst Platform Operational Analytics

If you used IBM SmartCloud Analytics Embedded in previous versions of IBM MobileFirst Platform Foundation for iOS, you must now migrate to IBM MobileFirst Platform Operational Analytics.

About this task

In IBM MobileFirst Platform Foundation for iOS V7.1.0, IBM MobileFirst Platform Operational Analytics replaces IBM SmartCloud Analytics Embedded. Complete the following steps to migrate to IBM MobileFirst Platform Operational Analytics. For more information about IBM MobileFirst Platform Operational Analytics, see "Operational analytics" on page 13-6.

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 MobileFirst Platform Operational Analytics" on page 6-191.
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.

Migration to the V7.1.0 Analytics Console

Starting with V7.1.0, the MobileFirst Analytics Console was improved, which requires migration of your analytics data.

Why you must migrate

In V7.1.0, the internals of the Analytics Console were improved, which required changing the format in which the data is stored. To continue to interact with the analytics data that was already collected, the data must be migrated into the new data format.

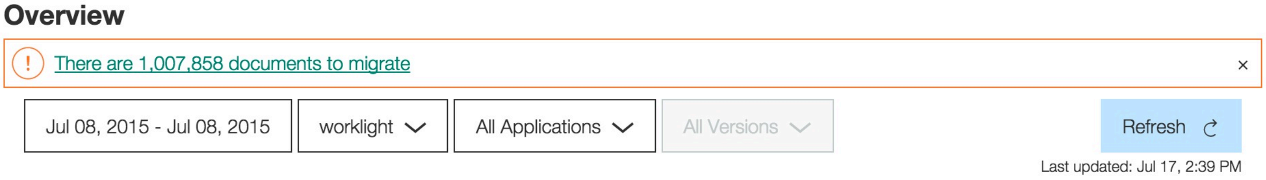
Viewing the Analytics Console for the first time after you upgrade

When you first view the Analytics Console after you upgrade to V7.1.0, no statistics are rendered in the console. Your data is not lost, but it must be migrated to the new data format.

How to migrate

An alert is displayed in the console that reminds you that there are documents to migrate. The alert includes a link to the Migration tab of the Administration page.

The following image shows a sample alert:



Overview

There are 1,007,858 documents to migrate

Jul 08, 2015 - Jul 08, 2015

worklight

All Applications

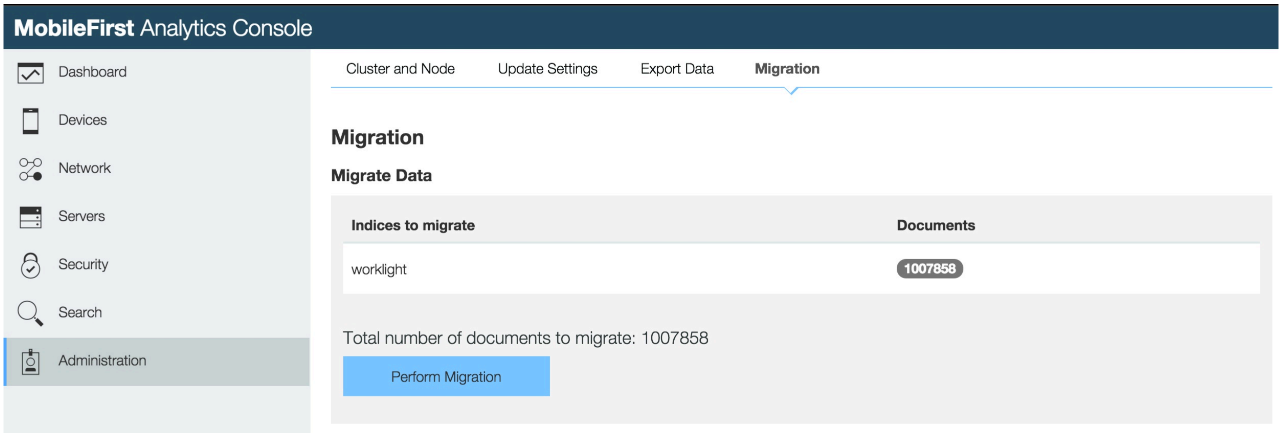
All Versions

Last updated: Jul 17, 2:39 PM

Migration tab

On the Migration tab of the Administration page, you can see how many documents must be migrated, and which indexes they are stored on. There is only one action: **Perform Migration**.

The following image shows the Migration tab of the Administration page:



Migration

Migrate Data

Indices to migrate	Documents
worklight	1007858

Total number of documents to migrate: 1007858

Note: This process might take a long time, depending on the amount of data you have, and it cannot be stopped during migration.

The migration takes approximately 3 minutes to migrate 1 million documents on a single node with 32G of RAM, with 16G allocated to the JVM, with a 4 core processor. Documents that are not migrated are not queried, and therefore are not rendered in the console.

Troubleshooting

If the migration fails while in progress, retry the migration. Retrying the migration does not remigrate documents that were already migrated, and your data integrity is maintained.

Upgrading to MobileFirst Server V7.1.0 in a production environment

Upgrading MobileFirst 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 V7.1.0 in a production environment, the process can be more complicated than upgrading to a new version in your development environment. The upgrade procedure can also take longer if you have existing MobileFirst applications that run in a production MobileFirst Server environment. For step-by-step instructions on how to upgrade your production MobileFirst Server to V7.1.0, 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 MobileFirst projects and applications to the new version, and performing other preparation tasks that must be completed before you install the new version of MobileFirst Server. These preparatory steps are followed by postinstallation, verification, and configuration tasks that must be completed before you restart the new MobileFirst Server and finish migrating your MobileFirst 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 MobileFirst Server V7.1.0 process," and then read through the steps under each of the major topics that follow.

Alternatively, you can refer to one of the following topics that provide the complete steps-by-steps instructions to guide you through the entire upgrading process of your MobileFirst Server to V7.1.0:

- Applying a fix pack for MobileFirst Server V7.1.0
- Upgrading MobileFirst Server from V7.0.0 to V7.1.0
- Upgrading MobileFirst Server from V6.3.0 to V7.1.0
- Upgrading MobileFirst Server from V6.2.0 to V7.1.0
- Upgrading MobileFirst Server from V6.1.0 to V7.1.0
- Upgrading MobileFirst Server from V6.0.0 to V7.1.0
- Upgrading MobileFirst Server from V5.0.6 to V7.1.0

Overview of the upgrade to MobileFirst Server V7.1.0 process

An overview of the MobileFirst Server V7.1.0 upgrade process, including what is updated and what is not.

A typical instance of MobileFirst Server includes the following elements:

- A database management system (DBMS) that runs databases for the Application Center and for MobileFirst Server. This DBMS hosts and run the following databases:

  - The Application Center database (if Application Center is installed on that server).
  - The administration database.
  - One or more runtime databases. Each runtime environment requires one runtime database, and an optional reports database.

    Note: The reports database is deprecated since IBM MobileFirst Platform Foundation for iOS V7.1.0. Use MobileFirst Operational Analytics instead. For more information, see "Installing the IBM MobileFirst Platform Operational Analytics" on page 6-191.
- 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 MobileFirst Operations Console application. One MobileFirst Operations Console can be used to administer several MobileFirst 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 MobileFirst Operations 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 MobileFirst runtime environments. Each MobileFirst runtime environment:
    - Is defined by a WAR file that is created with the MobileFirst Platform Command Line Interface for iOS development tool.
    - Is connected to two databases, one for runtime and one for reports.
    - Can run on one or more physical servers, for both workload and service availability considerations.
- An installation of the MobileFirst Server programs, usually on the same computer as the application server or deployment manager.

An IBM MobileFirst Platform Foundation for iOS installation might contain other elements, such as the MobileFirst Data Proxy, or the IBM MobileFirst Platform Operational Analytics. The upgrade procedures for these components are described in different sections.

The topics in this section focus on the task of upgrading and configuring the following entities:

- The MobileFirst Server programs.
- The databases, including the creation of the administration database.
- The MobileFirst project runtime applications and Application Center applications that are deployed in the application server.

  Note: The upgrade of the MobileFirst project runtime applications includes the installation and setup of the MobileFirst Operations Console and administration services applications.
- To upgrade the MobileFirst Data Proxy, see "Upgrading, or applying a fix pack to the MobileFirst Data Proxy" on page 7-84.
- To upgrade the MobileFirst Operational Analytics, see "Applying a fix pack to IBM MobileFirst Platform Operational Analytics" on page 7-85.

The actual steps that you must complete for the upgrade can change, depending on the particular *upgrade path* that you are pursuing. Your upgrade path is determined by your upgrading mode:

- From a previous version of Worklight Server or MobileFirst Server to MobileFirst Server V7.1.0 (for example, from V6.0.0.x to V7.1.0 or from V6.1.0.x to V7.1.0).
- From MobileFirst Server V7.1.0 to a fix pack release or an interim fix (for example, from V7.1.0 to V7.1.0.x).

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 MobileFirst upgrade path.
- Whether your Application Center, and MobileFirst Server (old version), are 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: MobileFirst 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. Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-3. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes/No
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes/No
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes/No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes/No
MobileFirst Server V6.3.0 to MobileFirst Server V7.1.0	Yes/No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes/No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes/No

Table 7-4. System status after this step.

Module	Status
Application Center	Running/Stopped/Uninstalled/Upgraded
MobileFirst Server	Running/Stopped/Uninstalled/Upgraded

Preparation for upgrades to MobileFirst Server

Before you begin the actual upgrade to MobileFirst Server V7.1.0, you must complete several preparation tasks.

Upgrading to a new version of MobileFirst Server in a development environment is quick and easy because in most cases no critical data must be preserved in IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Server V7.1.0 upgrade process" on page 7-33.

Gathering information for MobileFirst Server V7.1.0 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 MobileFirst 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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-5. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-6. System status after this step.

Module	Status
Application Center	All instances are kept running.

Table 7-6. System status after this step (continued).

Module	Status
MobileFirst Server	All instances are kept running.

Procedure

Go through the following checklist.

- Verify that your operating system, application server, and Database Management System (DBMS) meet the system requirements for MobileFirst Server V7.1.0 at Detailed System Requirements for IBM MobileFirst Platform Foundation for iOS 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 MobileFirst runtime databases.
- Gather the credentials to the MobileFirst Server administration, the MobileFirst runtime environments, the MobileFirst 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 administration database does not exist for IBM Worklight V6.1.0 or earlier.
- During the upgrade procedure, you must stop and restart the application server, and verify its configuration. Therefore, you must be familiar enough with your application server to complete these tasks, or have a system administrator do them.
- If the URL to the Application Center or the MobileFirst Server applications or their console changes, identify all the systems that you must update. If you upgrade from V6.1.0 to V7.1.0, the URLs to the Application Center and the MobileFirst runtime environment do not change, but a new URL is introduced for the MobileFirst Operations Console.
- The reports database is deprecated since IBM MobileFirst Platform Foundation for iOS V7.1.0. If you still use the reports database, you might consider using MobileFirst Operational Analytics instead. For more information, see "Operational analytics" on page 13-6.

  For installation instructions, see "Installing the IBM MobileFirst Platform Operational Analytics" on page 6-191, and "Setting up a production cluster" on page 13-85.

Planning installation of the MobileFirst Administration Services and MobileFirst Operations Console

You must plan the steps that you perform later to upgrade the Administration Services and the MobileFirst Operations Console. These components were 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 introduced a new architecture for the unified console based on several core elements that are described in "Introduction to the MobileFirst Server components" on page 6-6.

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 MobileFirst Operations 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 MobileFirst Operations Console, is at "Installation or upgrade of MobileFirst Server Administration Services" on page 7-49.

Note: If your application server is WebSphere Application Server V7.0, you must add the configuration property "`com.ibm.ws.webcontainer.invokerequestlistenerforfilter = true`". For more information about how to add this property, see "Configuring WebSphere Application Server V7.0" on page 6-59.

About this task

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-7. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-8. System status after this step.

Module	Status
Application Center	All instances are kept running.
MobileFirst Server	All instances are kept 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 "Planning deployment of administration components and runtimes" on page 6-8.
2. Define your upgrade strategy if multiple MobileFirst runtimes are installed (project WAR files).
3. Prepare the configuration of the application.
4. Set up the MobileFirst 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 the MobileFirst Server administration" on page 6-47.

The following steps are for planning only, and you do not have to start the installation of the MobileFirst Server administration at this stage. The actual installation is described later in the upgrade procedure, at "Installation or upgrade of MobileFirst Server Administration Services" on page 7-49.

Procedure

1. Review the topic "Planning deployment of administration components and runtimes" on page 6-8 to define the application server where you can install the MobileFirst Server administration components.
2. Define your upgrade strategy if multiple MobileFirst runtimes are installed (project WAR files).

   Note: You must perform this step only if you have more than one MobileFirst runtimes (project WAR files) to upgrade. If you have only one MobileFirst runtime to upgrade, you can skip this test.

   You can either manage all the runtimes with the same MobileFirst Administration Services and Console runtime environment or install this environment for each runtime.

   - Manage all the runtimes with the same MobileFirst 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 MobileFirst administration data is migrated to the administration database, you can no longer change the context root of a MobileFirst Server runtime. For more information, see "Upgrade the runtime database" on page 7-52.
   - Install a MobileFirst 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 MobileFirst runtime environments" on page 15-32 and "Ant tasks for installation of MobileFirst Operations Console and Administration Services" on page 15-24.
     - If you install manually, update the **`ibm.worklight.admin.environmentid`** JNDI property as documented in "List of JNDI properties for MobileFirst Server administration" on page 6-101 and "JNDI environment entries for MobileFirst projects in production" on page 11-67.

3. Prepare the configuration of the application.

   Since IBM Worklight Foundation V6.2.0, the protection of the Administration Services and the MobileFirst Operations Console is configured by security roles that are managed in the application server. For more information, see "Configuring user authentication for MobileFirst Server administration" on page 6-96. To prepare the installation and configuration of the Administration Services and the MobileFirst Operations 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 MobileFirst Operations Console when the applications are installed.
4. Set up the MobileFirst administration database.

   A MobileFirst 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 6-47.
5. Review the configuration of the application server.

   For IBM MobileFirst Platform Foundation for iOS V7.1.0, you must configure your application server to enable Java Management Extensions (JMX) communication between the Administration Services and the MobileFirst Server runtime. Review the topic "Configuration of the application server" on page 6-52 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 MobileFirst WAR file and prepare the Ant deployment script

In this task, you identify the MobileFirst project WAR file that contains numerous resources and configuration settings for MobileFirst Server and prepare the Ant script that is used to deploy it.

About this task

The MobileFirst project WAR file, also referred to as the MobileFirst runtime environment, is a web application archive that contains server-specific configuration settings, and other resources that can be required to run MobileFirst applications and adapters. You produce this file with MobileFirst Studio, or IBM MobileFirst Platform Command Line Interface for iOS, and it is the end point for mobile applications. Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-9. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes

Table 7-9. Is this step required for your upgrade path? (continued).

Product or component versions	**Required?**
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-10. System status after this step.

Module	**Status**
Application Center	Running
MobileFirst Server	Running

Procedure

In the upgrade process, the MobileFirst runtime environment must be migrated, and redeployed to the application server. It is important to deploy the same WAR file than the one you previously deployed to your production server. This WAR file might have been updated since the last installation of MobileFirst Server, or the last upgrade. 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 MobileFirst applications on the server" on page 11-50.

     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 11-5.
   - When you upgrade from Worklight Server V5.0.6.x, a MobileFirst 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 11-4.

     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 migrated to MobileFirst Server V7.1.0 format during the deployment procedure that is described in later steps.
2. Prepare the Ant deployment script that you use to upgrade this WAR file to MobileFirst Server V7.1.0 and to deploy this WAR file to the application server, with the upgraded MobileFirst runtime library.

a. Depending on the version from which you want to upgrade to V7.1.0, you must select one of the following options to prepare your Ant tasks.

   - Use the Ant samples provided in the `WorklightServer/configuration-samples/` directory of the MobileFirst Server V7.1.0 distribution. Compared to previous versions, these Ant tasks do not install or upgrade the deprecated reports database. This method is preferred when you upgrade from V5.0.6.x, V6.0.0 or V6.1.0, when you do not want to upgrade your reports database, or if you cannot find the Ant tasks you used to install previous versions.
   - Reuse the Ant files that you used to install previous versions. The Ant files provided by previous versions do install and upgrade the reports database. This method is preferred when you upgrade from V6.2.0, V6.3.0, or V7.0.0, and you want to keep you reports database. You cannot use this method if you upgrade from V6.0.0 or V6.1.0, because the Ant tasks that are provided in these versions are not compatible with MobileFirst Server V7.1.0.

     If you upgrade from V6.0.0 to V6.1.0, you can encounter the following incompatibilities: The name of the `worklight-ant-deployer.jar` file is modified (`taskdef` element), and new targets are added to perform an upgrade without losing configuration (`minimal-update`).

     If you upgrade from V6.1.0 to V6.2.0, you can encounter the following incompatibilities: New targets are added to install the MobileFirst Administration Services, the MobileFirst Operations Console, and their databases. A new element is added to the `configuredatabase` task.

b. To use a sample ant file that is provided in the `WorklightServer/configuration-samples/` directory:

   1) You must install MobileFirst Server V7.1.0 on your computer, without installing Application Center.
   2) Review the sample configuration files in “Sample configuration files” on page 15-66, and copy the Ant sample file that corresponds to your application server and your database. For more information, see the table Table 15-83 on page 15-67.
   3) There are two different cases, depending on the version to be upgraded:
      - For an upgrade from V6.0.0 and later, the files for upgrading your runtime and your database are named after the following pattern:

        `configure-appServer-database.xml`
      - For an upgrade from V5.0.6.x, the files for upgrading your runtime and your database are named after the following pattern:

        `redeploy506-appServer-database.xml`

        Note: These files have some parameters that are already entered to match the installation settings of IBM Installation Manager for IBM Worklight V5.0.6, especially for the database settings.

c. You find the following kind of properties in your configuration sample file:

```
<property name= "database.db2.wladmin.password" value= "************" />
<propertyname="database.db2.worklight.password"value="************"/>
<property name= "worklight.server.install.dir" value= "****UPDATE**** - MobileFirst Server
<property name= "worklight.project.war.file" value= "****UPDATE**** - MobileFirst Project W
<property name= "worklight.contextroot" value= "****UPDATE**** - Context Root for the Mobil
<property name= "database.db2.host" value= "****UPDATE**** - DB2 host name (example: proddb
<property name= "database.db2.port" value= "****UPDATE**** - DB2 port, by default 50000 (ex
<property name= "database.db2.instance" value= "****UPDATE**** - DB2 database instance (exa
<property name= "database.db2.driver.dir" value= "****UPDATE**** - DB2 JDBC driver director
<property name= "database.db2.wladmin.dbname" value= "****UPDATE**** - Database name for Mo
```

```
<property name= "database.db2.wladmin.schema" value= "****UPDATE**** - Database schema for Mob
<property name= "database.db2.wladmin.username" value= "****UPDATE**** - DB2 user name (examp
<property name= "database.db2.worklight.dbname" value= "****UPDATE**** - Database name for Mob
<property name= "database.db2.worklight.schema" value= "****UPDATE**** - Database schema for M
<property name= "database.db2.worklight.username" value= "****UPDATE**** - DB2 user name (exam
<property name= "appserver.was.installdir" value= "****UPDATE**** - Installation directory of
<property name= "appserver.was85liberty.serverInstance" value= "****UPDATE**** - The name of t
```

You must specify precisely the values of these properties. For example, for the property named "worklight.server.install.dir", you must replace the following string with the actual directory where you installed MobileFirst Server:

```
"****UPDATE**** - MobileFirst Server installation directory (example: /opt/IBM/MobileFirst_Pla
```

For passwords, such as "database.db2.wladmin.password", if you do not specify them in the Ant sample, then you are asked to enter them when it is required by the upgrading process.

Note: The properties in the previous example are for WebSphere Application Server Liberty profile, and DB2 database. For a different application server, or a different database, you have different properties to set.

If you are not familiar with the MobileFirst component, the meaning of the properties is as follows:

- the MobileFirst project WAR file is a WAR file that is built by your development team with MobileFirst Studio, or with MobileFirst Platform Command Line Interface for iOS.
- The database for the Administration Services is a database that was introduced in V6.2.0, which contains applications, adapters, and other administration data. If you upgrade from V6.2.0 or later, you must use the same parameters as the ones you used at installation time, so that the upgrade is performed. If you upgrade from V6.1.0 or earlier, a new database must be created. For more information about the creation of this database, see “Installation or upgrade of MobileFirst Server Administration Services” on page 7-49.
- The database for the MobileFirst runtime environment is a database used by the MobileFirst project WAR file. Use the same parameters as the ones you used at installation time, so that the upgrade is performed.

Since V7.1.0, the Ant samples that are provided do not upgrade your reports database. If you still need to keep your reports database, and you want to upgrade it to V7.1.0, you must do extra customization of the Ant sample, in addition to the modification performed in this step. For more information, see “Using Ant tasks to upgrade the reports database” on page 13-121.

3. Verify that the **environmentID** attribute for the MobileFirst runtime environments matches the **environmentID** attribute that is used to install the MobileFirst Server administration Ant file.

   If you install the MobileFirst Server administration components with a different Ant file than the one that you used to install the MobileFirst runtime environment, for example if you install the MobileFirst Server administration with the Server Configuration Tool, you might have a different **environmentId** for the administration and the runtime. In this case, the MobileFirst Server administration components would not find the MobileFirst 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 MobileFirst Operations Console and Administration Services” on page 15-24.
- `configureapplicationserver`, `updateapplicationserver`, `unconfigureapplicationserver` Ant tasks, which are documented at “Ant tasks for installation of MobileFirst runtime environments” on page 15-32.

Review and note the application server configuration for MobileFirst 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 Console and Application Center Services
- For Worklight Server or MobileFirst Server:
  - The Administration Console and Administration Services
  - Each project runtime

The JDBC data sources to review are as follows:

- For Application Center: the Application Center database
- For Worklight Server or MobileFirst Server:
  - The runtime database
  - The reports database
  - The administration database

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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-11. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes

Table 7-11. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-12. System status after this step.

Module	Status
Application Center	All instances are kept running.
MobileFirst Server	All instances are kept 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 MobileFirst Server V7.1.0, verify that all of the environments that are targeted in your MobileFirst applications are still supported.

About this task

After the migration is completed, your MobileFirst applications contain only the environments that are supported by the current version of MobileFirst Server.

Since IBM Worklight V6.1.0, some of the MobileFirst 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 MobileFirst Server prior to V7.1.0, the application on this device stops working after an upgrade of MobileFirst Server to V7.1.0.

Therefore, if you upgrade Worklight Server from V6.0.0.x or earlier to MobileFirst Server V7.1.0, you must pay particular attention to the following procedure. Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-13. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-14. System status after this step.

Module	Status
Application Center	Running
MobileFirst Server	Running

Procedure

If your current version of MobileFirst Server includes existing applications that target environments that are no longer supported by MobileFirst Server V7.1.0:

- For old, no-longer-supported environments, your application developers must update the MobileFirst application to run with an environment supported by MobileFirst Server V7.1.0 before you can run it.
- For new environments for which support is added after the release of MobileFirst Server V7.1.0, check for the availability of a fix pack release that provides support for this environment.

The following table can help to determine the IBM Worklight versions in which support for older environments was discontinued, and to suggest possible replacement environments for those environments.

Environment	Support removed in	Suggested replacement path
Facebook	IBM Worklight V6.0.0	Desktop web app
iGoogle	IBM Worklight V6.0.0	Review environments supported by IBM MobileFirst Platform Foundation for iOS V7.1.0

Environment	Support removed in	Suggested replacement path
Apple OS X Dashboard	IBM Worklight V6.0.0	Review environments supported by IBM MobileFirst Platform Foundation for iOS V7.1.0
Windows 7 and Vista	IBM Worklight V6.0.0	Review environments supported by IBM MobileFirst Platform Foundation for iOS V7.1.0
Windows Phone 7.5	IBM Worklight V6.1.0	Review environments supported by IBM MobileFirst Platform Foundation for iOS V7.1.0

Related concepts:

"Version compatibility" on page 7-1
Compatibility among different versions of the IBM MobileFirst Platform Foundation for iOS client and server depends on several factors.

In-place upgrade or rolling upgrade to MobileFirst Server V7.1.0

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 or MobileFirst Server is no longer installed after the new version of MobileFirst Server is installed.

In-place upgrades require some downtime of the service.

Rolling upgrade

An upgrade that installs the new version of MobileFirst Server such that it runs side-by-side with the old version of Worklight Server or MobileFirst 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 MobileFirst Platform Foundation for iOS V7.1.0 is documented in "Rolling upgrade procedure to apply a fix pack to IBM MobileFirst Platform Foundation for iOS V7.1.0 in stateful mode" on page 7-74.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-15. Is this step required for your upgrade path?

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.0 to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.0 to MobileFirst Server V7.1.0.x	Yes

Table 7-15. Is this step required for your upgrade path? (continued)

Product or component versions	Required?
Worklight Server V6.2.0 to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.0 to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-16. System status after this step

Module	Status
Application Center	Running
MobileFirst Server	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 MobileFirst Server.

About this task

Important: The information on this page applies to you if you previously installed Worklight Server V5.x with the embedded WebSphere Application Server Liberty profile option.

Since IBM Worklight V6.1.0, WebSphere Application Server Liberty Core is not embedded in the IBM Installation Manager wizard of MobileFirst Server. Instead, it is provided as a separate IBM Installation Manager wizard.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-17. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	No
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-18. System status after this step.

Module	Status
Application Center	All instances are kept running.
MobileFirst Server	All instances are kept running.

Procedure

As a result of this packaging, the MobileFirst 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 MobileFirst 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 MobileFirst apps and data to a new Liberty server

This alternate upgrade method migrates your MobileFirst Operations 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 MobileFirst Platform Foundation for iOS.
2. Install WebSphere Application Server Liberty Core with IBM Installation Manager. The installer for WebSphere Application Server Liberty Core is part of the IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Server” on page 6-24.
4. Configure the Liberty server for your production environment.
5. Modify the Ant files created in section “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21 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 7-40, follow the instructions for “Upgrading from Worklight Server V5.0.6.x (changing the Liberty server)” on page 7-44.

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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-19. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-20. System status after this step.

Module	Status
Application Center	Running
MobileFirst Server	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.

   Verify that your version of IBM Installation Manager is supported by IBM MobileFirst Platform Foundation for iOS V7.1.0. For more information, see the prerequisite software in “System requirements” on page 2-9. For more information about IBM Installation Manager procedures, see the IBM Installation Manager user documentation.

   IBM Installation Manager updates are available at Installation Manager and Packaging Utility download links.

   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 or MobileFirst 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 6-34 and “Working with sample response files for IBM Installation Manager” on page 6-35 and prepare your response file.

   To prepare your response file from sample response files, create a response file based on the following versions of MobileFirst Server, and sample files:

   Table 7-21. Sample upgrade response files in the `Silent_Install_Sample_Files.zip`

Initial version of MobileFirst Server	Sample file
Worklight Server V5.x	`upgrade-initially-worklightv5.xml`
Worklight Server V6.x	`upgrade-initially-worklightv6.xml`
IBM MobileFirst Platform Server V6.x	`upgrade-initially-mfpserverv6.xml`
IBM MobileFirst Platform Server V7.x	`upgrade-initially-mfpserver.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, MobileFirst Server V7.1.0), you need only:

- The base repository V7.1.0 installation package from Passport Advantage or physical media.

To install a fix pack release (for example, MobileFirst Server V7.1.0.1), you need:

- The corresponding base repository (such as MobileFirst Server V7.1.0) installation package from Passport Advantage or physical media. The corresponding base repository for V7.1.0.x fix packs is the V7.1.0 release.
- The appropriate V7.1.0.x installation package from FixCentral.

For a fix pack installation, you must add both repositories to the list known to IBM Installation Manager. Then, in the example given, IBM Installation Manager recognizes the V7.1.0 release as an **Install** choice and the V7.1.0.x release (or interim fix) as an **Update** choice.

To install an interim fix release, you can need up to three repositories:

- The repositories for the release to which the fix applies.
- The repository for the fix.

For installing an interim fix, you must add all these repositories to the list known to IBM Installation Manager. Then IBM Installation Manager recognizes the interim fix as an Update choice.

Review of the basic IBM Installation Manager steps

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 MobileFirst Server V7.1.0 upgrade process

In this phase of the upgrade process, you shut down and back up the application server and MobileFirst databases and perform other pre-installation tasks.

When you finish the tasks that are listed in "Preparation for upgrades to MobileFirst Server" on page 7-16, you can begin the actual upgrade process.

Note: After you complete this phase of the upgrade process, your MobileFirst 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 MobileFirst Server files

Before you begin the actual installation, check the ownership of all MobileFirst Server files.

About this task

The upcoming step "Running IBM Installation Manager and completing the Application Center upgrade" on page 7-40 attempts to remove and replace many files in theMobileFirst 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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-22. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-23. System status after this step.

Module	Status
Application Center	All instances are kept running.
MobileFirst Server	All instances are kept running.

Procedure

If you previously installed MobileFirst Server with the nonadministrator (single-user) mode of IBM Installation Manager, check whether all files and directories in the `product_install_dir` installation directory are owned by the current user.

For more information about Installation Manager administrator and nonadministrator modes, see Administrator, nonadministrator, and group mode. Group mode is not supported for MobileFirst Server installation.

On UNIX, you can use the following command to list the files and directories that do not fulfill this condition.

```
cd product_install_dir
find . '!' -user "$USER" -print
```

This command is expected to return nothing.

What to do next

See also: “File system prerequisites” on page 6-5

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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-24. Is this step required for your upgrade path?.

Product or component versions	**Required?**
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-25. System status after this step.

Module	**Status**
Application Center	All instances are running.
MobileFirst Server	All instances are running.

Procedure

Back up all the application servers (or network deployment nodes) where the Application Center and MobileFirst 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 upgrade from IBM Worklight V5.0.6, or if you installed Application Center with IBM Installation Manager, and if you use WebSphere Application Server Liberty profile or Apache Tomcat, you must shut down the application server.

About this task

When it is running, IBM Installation Manager removes the Worklight application from the application server if you upgrade from V5.0.6. It also updates Application Center if you installed Application Center with IBM Installation Manager. If you upgrade from V5.0.6, or if you installed Application Center with Installation Manager, 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 MobileFirst installation directory hierarchy.
  - On Windows, you must also shut down all such processes inside the Liberty MobileFirst 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 MobileFirst Server installation in an inconsistent state.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-26. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes (if you installed Application Center with IBM Installation Manager)
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)

Table 7-26. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes (if you installed Application Center with IBM Installation Manager)

Table 7-27. System status after this step.

Module	Status
Application Center	Liberty and Tomcat are stopped. All other instances are kept running.
MobileFirst Server	Liberty and Tomcat are stopped. All other instances are kept running.

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 MobileFirst Server installation.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-28. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes (if installed on multiple servers)
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes (if installed on multiple servers and if you installed Application Center with IBM Installation Manager)

Table 7-29. System status after this step.

Module	Status
Application Center	All instances are stopped.
MobileFirst Server	Liberty and Tomcat are stopped. All other instances are kept running.

Procedure

This step is required because IBM Installation Manager migrates the schema of the database so that it can be used with MobileFirst Server V7.1.0. No instance of Application Center can be running while this operation is performed.

After the database is migrated, only migrated Application Center applications 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 MobileFirst Server V7.1.0, back up your Application Center database.

About this task

In the upgrade process, the Application Center database is updated and migrated to a schema compatible with MobileFirst Server V7.1.0. This operation cannot be undone. If, for any reason, you decide to roll back the upgrade of MobileFirst Server, you need this backup.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-30. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes (if you installed Application Center with IBM Installation Manager)
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes (if you installed Application Center with IBM Installation Manager)
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-31. System status after this step.

Module	Status
Application Center	All instances are stopped.
MobileFirst Server	Liberty and Tomcat are stopped. The other instances keep running.

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 runtime and reports databases are backed up as well, but in a later step of this procedure. For more information, see step "Back up the runtime database" on page 7-50 of this upgrade procedure.

Running IBM Installation Manager and completing the Application Center upgrade

Use IBM Installation Manager to install the new MobileFirst Server version.

Before you continue, make sure that you completed all of the steps in the “Preparation for upgrades to MobileFirst Server” on page 7-16 and “Starting the MobileFirst Server V7.1.0 upgrade process” on page 7-33 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 MobileFirst Platform Foundation for iOS. For more information, see “Command-line installation with XML response files (silent installation)” on page 6-34 and “Working with sample response files for IBM Installation Manager” on page 6-35.

Upgrading from MobileFirst Server V6.3.0, or later

In this step, you run IBM Installation Manager to perform the upgrade to MobileFirst Server V7.1.0.

About this task

IBM Installation Manager completes the following tasks:

- It installs on your disk the files and tools that are required to deploy IBM MobileFirst Platform Foundation for iOS on your application server.
- If Application Center was installed in the previous version of MobileFirst 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 IBM MobileFirst Platform Foundation for iOS V7.1.0. To see a copy of the upgrade scripts, you can install MobileFirst Server in a new package group and review a copy of the upgrade scripts in `<product_install_dir>/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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-32. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	No
Worklight Server V6.0.0.x to MobileFirst Server V7.1.0	No
Worklight Server V6.1.0.x to MobileFirst Server V7.1.0.x	No
Worklight Server V6.2.0.x to MobileFirst Server V7.1.0	No

Table 7-32. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
MobileFirst Server V6.3.0.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-33. System status after this step.

Module	Status
Application Center	All instances are stopped.
MobileFirst Server	Liberty and Tomcat are stopped. All other instances are kept running.

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 V6.0.0, V6.1.0, or V6.2.0

In this step, you run IBM Installation Manager to perform the actual upgrade from IBM Worklight V6.x to MobileFirst Server V7.1.0.

About this task

IBM Installation Manager completes the following tasks:

- It installs on your disk the files and tools that are required to deploy IBM MobileFirst Platform Foundation for iOS on your application server.
- If Application Center was installed in the previous version of IBM Worklight, 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 MobileFirst Platform Foundation for iOS V7.1.0. To see a copy of the upgrade scripts, you can install MobileFirst Server in a new package group and review a copy of the upgrade scripts in *product_install_dir*/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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-34. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	No
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-35. System status after this step.

Module	Status
Application Center	Stopped (all instances
MobileFirst Server	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 **Install**.

   Note: In IBM MobileFirst Platform Foundation for iOS, you must upgrade by clicking **Install**, because the package name for Worklight Server changed between Worklight Server and MobileFirst Server.
3. Select the package group that contains your Worklight Server installation.
4. 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 MobileFirst Server V7.1.0 in a stand-alone WebSphere Application Server or Apache Tomcat environment.

About this task

If you originally installed IBM Worklight on the following application servers, use the IBM Installation Manager **Install** function as explained in the steps:

- A stand-alone WebSphere Application Server Liberty profile server
- A stand-alone WebSphere Application Server full profile server
- A stand-alone Apache Tomcat server

Read from the tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-36. Is this step required for your upgrade path?.

Product or component versions	**Required?**
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Worklight Server V5.0.6.x to MobileFirst Server V7.1.0
Worklight Server V6.0.x to MobileFirst Server V7.1.0	No
Worklight Server V6.1.x to MobileFirst Server V7.1.0	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-37. System status after this step.

Module	**Status**
Application Center	All instances are stopped.
MobileFirst Server	Uninstalled.

Procedure

1. Start IBM Installation Manager.
2. Click **Install**. The package name for MobileFirst Server has changed between Worklight Server V5.x and MobileFirst Server V7.1.0, 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 MobileFirst Server V7.1.0” on page 7-28), 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 IBM MobileFirst Platform Foundation for iOS in your application server.
   - It undeploys the previous version of IBM 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 MobileFirst Server. To see a copy of the upgrade scripts, you can install MobileFirst Server in a new package group and review a copy of the upgrade scripts in *<product_install_dir>*/ `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 MobileFirst apps and data to a new Liberty server" on page 7-30 section of the "Packaging change of WebSphere Application Server Liberty profile in IBM Worklight V6.x" on page 7-29 step. Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-38. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes (if Liberty server was installed by Worklight Server V5.0.6)
Worklight Server V6.0.x to MobileFirst Server V7.1.0	No
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-39. System status after this step.

Module	Status
Application Center	Stopped (all instances)
MobileFirst Server	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 MobileFirst Server and Application Center" on page 7-25.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-40. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-41. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	After an upgrade from V6.0.x or later, all instances are running. After an upgrade from V5.0.6.x, all instances are stopped.

Procedure

1. For the applications, restore the Application Center console and services.
2. For the JDBC data sources, restore 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 MobileFirst Operations Console, that application server is shut down again in a later step, and is restarted only in subsequent steps.

Upgrading the MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks

In these post-installation steps, you upgrade the MobileFirst Server, its databases, its MobileFirst runtime environment or environments, and restart the application server.

This topics in this section describe the steps to upgrade MobileFirst Server and MobileFirst runtime environments with Ant tasks, or manually. If you installed MobileFirst Server and the MobileFirst runtime environments with the Server Configuration Tool, you can follow the procedure "Upgrading with the Server Configuration Tool" on page 7-72 instead.

Complete each of the following steps, as required for your particular upgrade path.

Stop all MobileFirst Server instances

Before you complete subsequent upgrade steps, you must shut down all 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 or MobileFirst Server on multiple servers, whether networked or not, you must stop all runtime environments before you move on to the next steps.

Note: You must do so even if you installed only one runtime environment.

This step is mandatory because in step "Upgrade the runtime database" on page 7-52, you upgrade the schema of the databases so that it can be used with MobileFirst Server V7.1.0. No database schema can be upgraded while a runtime environment is running.

After the database is upgraded, only upgraded runtime environments can run, because only upgraded applications can read and write to the new databases. Otherwise, the database might be corrupted.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-42. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes

Table 7-42. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

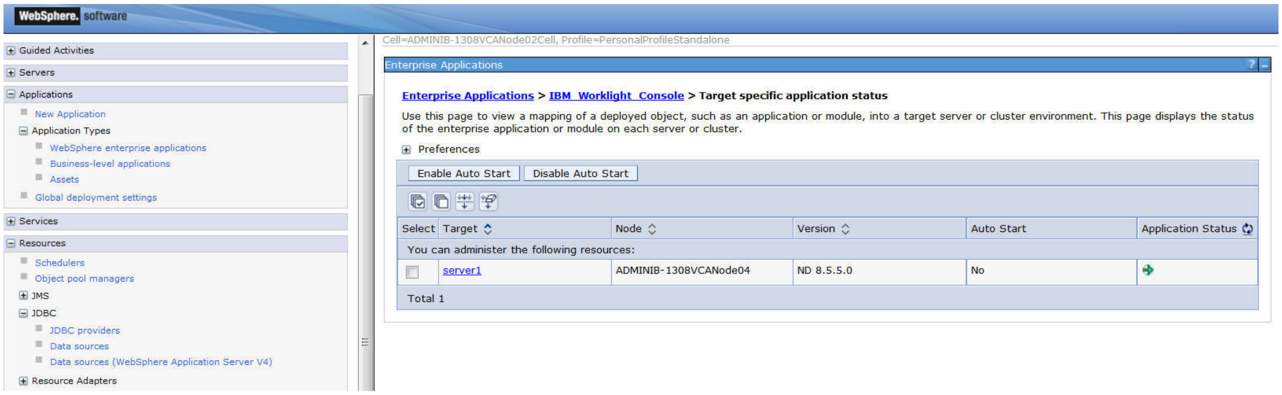
Table 7-43. System status after this step.

Module	Status
Application Center	All instances are upgraded.
MobileFirst Server	All instances are stopped.

Important: In addition to stopping all runtime environments, if you upgrade from IBM Worklight V6.0.0.x on WebSphere Application Server full profile, you must also 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 MobileFirst runtime environment for MobileFirst Server V7.1.0 manually, or with Ant tasks” on page 7-46 and prevents the new MobileFirst 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 **Target Specific Application Status**.
5. Select all the target servers, or the cluster where the application is installed.
6. Click **Disable Auto Start**.
7. Click **Save** to save the configuration.
8. Verify that the **Auto Start** property in the table is set to **No**.



Cell=ADMINIB-1308VCANode02Cell, Profile=PersonalProfileStandalone

Enterprise Applications > IBM_Worklight_Console > Target specific application status

Use this page to view a mapping of a deployed object, such as an application or module, into a target server or cluster environment. This page displays the status of the enterprise application or module on each server or cluster.

Select	Target	Node	Version	Auto Start	Application Status
You can administer the following resources:					
□	server1	ADMINIB-1308VCANode04	ND 8.5.5.0	No	➜
Total 1					

Figure 7-2.

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" on page 7-49 section), you must shut down the application server before proceeding to subsequent steps.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-44. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-45. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	All instances are stopped.

Procedure

In the following cases, you must shut down the application server before you undeploy applications from the MobileFirst Operations 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 MobileFirst Server Administration Services

As part of the MobileFirst Server upgrade, you must install the Administration Services, and optionally the MobileFirst Operations Console.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-46. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-47. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	All instances are stopped.

The procedure is different depending on whether you upgrade from a previous version to V6.2.0, or from a previous version to V7.1.0, or to a fix pack or interim fix.

For an upgrade from Worklight Server V6.1.0 or earlier to MobileFirst Server V7.1.0
Follow the steps in “Installing the MobileFirst Server administration” on page 6-47.

For an upgrade from Worklight Server V6.2.0, or later, to MobileFirst Server V7.1.0 (but not for an upgrade from MobileFirst Server V7.1.0 to a fix pack)

1. Back up the administration database
2. Find the Ant file that you created in "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21.
3. 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 MobileFirst Server V7.1.0.
4. Upgrade the administration database:

   ```
   product_install_dir/shortcuts/ant -f your_file admdatabases
   ```

   Note: For a manual upgrade of the administration database, instead of running the Ant tasks, see "Manually upgrading the MobileFirst Server V7.1.0 databases" on page 7-62.
5. Run the **`minimal-admupdate`** target of the Ant file:

   ```
   product_install_dir/shortcuts/ant -f your_file minimal-admupdate
   ```

   Note: For a manual upgrade of the applications server, instead of running the Ant task, review the manual installation instructions at "Deploying the Administration Services and MobileFirst Operations Console and configuring the application server manually" on page 6-81, and update the installed WAR files for `worklightconsole.war`, and `worklightadmin.war` in your installation.

For an upgrade from MobileFirst Server V7.1.0 to a fix pack or to an interim fix

1. Find the Ant file that you created in "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21.
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 MobileFirst Server V7.1.0.
3. Run the **`minimal-admupdate`** target of the Ant file:

   ```
   product_install_dir/shortcuts/ant -f your_file minimal-admupdate
   ```

   Note: For a manual upgrade of the applications server, instead of running the Ant task, review the manual installation instructions at "Deploying the Administration Services and MobileFirst Operations Console and configuring the application server manually" on page 6-81, and update the installed WAR files for `worklightconsole.war`, and `worklightadmin.war` in your installation.

Back up the runtime database

Back up the contents of your MobileFirst project database.

About this task

Important: Before performing this step, verify that you have completed step "Stop all MobileFirst Server instances" on page 7-46, and that no instance of MobileFirst Server is still running, and thus still using this database.

If you upgrade from IBM Worklight V6.1.0 or earlier, during the upgrade process in the steps "Upgrade the runtime database" on page 7-52 and "Upgrade the MobileFirst Server runtime environment" on page 7-54, the data that is specific to administration and runtime environments are split into distinct databases:

- The MobileFirst data that is related to administration is moved to the administration database.

- The runtime database that you previously updated is migrated to a schema compatible with MobileFirst Server V7.1.0.

If you upgrade from V6.2.0, or later, the database schema is modified and some data might be modified.

The previous operations cannot be undone.

If, for some reason, you decide to roll back the upgrade of MobileFirst Server, you need this backup.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-48. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-49. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are partially upgraded.

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`

Note: The reports database and the sample BIRT reports are deprecated since IBM MobileFirst Platform Foundation for iOS V7.0.0. Use “Operational analytics” on page 13-6 instead.

Upgrade the runtime database

If you upgrade from IBM Worklight V6.1.0 or earlier, you must move the data that is related to administration to the administration database. In all cases, you must also upgrade the runtime database to a schema that is compatible with MobileFirst Server V7.1.0.

Before you begin

1. Make sure that you complete step “Stop all MobileFirst Server instances” on page 7-46, and that no instance of Worklight Server or MobileFirst Server is still running, and therefore is still using these databases.
2. Make sure that you complete step “Installation or upgrade of MobileFirst Server Administration Services” on page 7-49 and that the administration database exists.

Note: This procedure explains how to upgrade the database with Ant tasks. For a manual upgrade of the databases, see “Manually upgrading the MobileFirst Server V7.1.0 databases” on page 7-62 instead.

Remember: To upgrade from V6.1.0 or earlier with Oracle as a database, the Oracle user must have the `CREATE VIEW` privilege.

About this task

In this step, you run Ant scripts to perform operations on your MobileFirst Server databases.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-50. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-51. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	All instances are stopped.

Procedure

1. If you upgrade from IBM Worklight V6.0.0.x, and the application server is WebSphere Application Server full profile, make sure that you disabled the auto start mode for all instances of the Worklight Console application, as specified in “Stop all MobileFirst Server instances” on page 7-46.
2. Locate the Ant file that you created in section “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21.
3. Verify that `taskdef` for the `worklight-ant-deployer.jar` uses the directory that contains the upgraded installation of MobileFirst Server V7.1.0.

   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 MobileFirst Platform Foundation for iOS that you use to upgrade the databases, to deploy the WAR file, and to install the MobileFirst runtime library for the MobileFirst Operations Console.
4. If you upgrade from IBM Worklight V6.1.0 or earlier, make sure that the Ant task `<configuredatabase kind ="Worklight">` in the Ant file contains an `<admindatabase>` sub-element.

   Note: In the following example code, DB2 is the DBMS. The **`${contextRoot}`** property contains the value of the context root of the MobileFirst project.

   ```
   <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>
```

5. Start the **databases** target of the Ant file with this command:

   `product_install_dir/shorcuts/ant -f your file databases`

   Note: If you created an Ant file with your own target names, the Ant task to start is **configuredatabase**.
6. 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 7-69.

This procedure loads the administration database and upgrades the database schemas for the runtime database to V7.1.0.

Upgrade the MobileFirst Server runtime environment

In this step, you run the Ant script to upgrade MobileFirst runtime environment to V7.1.0. You must repeat this procedure as many times for each runtime environment to upgrade.

Before you begin

Note: If your application server is WebSphere Application Server V7.0, you must add the configuration property "com.ibm.ws.webcontainer.invokerequestlistenerforfilter = true". For more information about how to add this property, see "Configuring WebSphere Application Server V7.0" on page 6-59.

About this task

Note: This procedure explains how to upgrade the applications that you deployed to the application server with Ant tasks. For a manual upgrade of the applications that you deployed in the application server, see "Manually upgrading the application server" on page 7-70.

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.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-52. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes

Table 7-52. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-53. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are partially upgraded.

Procedure

1. Locate the Ant file that you created in section “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21.
2. Verify that the `taskdef` for the `worklight-ant-deployer.jar` uses the correct directory containing the upgraded installation of MobileFirst Server V7.1.0.

   In this example, 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="${product_install_dir}/WorklightServer">
           <include name="worklight-ant-deployer.jar"/>
         </fileset>
       </classpath>
     </taskdef>
   ```

   This verification step is extremely important. It defines the version of IBM MobileFirst Platform Foundation for iOS that you use to migrate the databases, to deploy the WAR file, and to install the MobileFirst runtime library for the MobileFirst Operations Console.
3. Verify that the **`environmentID`** attribute for the MobileFirst runtime environment matches the **`environmentID`** attribute that you used to install the MobileFirst Server administration file. For more information, see “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21.

   Note:

   - If the **`environmentID`** is defined in the Administration Services, and not in the runtime, or if it has different values in each, the Administration Services application does not detect the runtime. The MobileFirst Operations Console would display “**No runtime found**”.

- If you installed the MobileFirst Server administration with the Server Configuration Tool, it might have added an **environmentID** attribute, whose value you can see in the **Configuration Details** of the Server Configuration Tool).

4. 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**, then **install**.
   - To upgrade from V6.2.0.x, or later (including from V7.1.0 to a fix pack), use **minimal-update**.
5. Run Ant with the selected target:

   `*product_install_dir*/shortcuts/ant -f *your file target defined in step 4*`

   This scripts have the following effects:
   - Target **uninstall**: see the definition for the task `<unconfigureapplicationserver>` in the *Task effects* section of the page Ant tasks for installation of MobileFirst runtime environments.
   - Target **install**: see the definition for the task `<configureapplicationserver>` in the *Task effects* section of the page Ant tasks for installation of MobileFirst runtime environments.
   - Target **minimal-update**: see the definition for the task `<updateapplicationserver>` in the *Task effects* section of the page Ant tasks for installation of MobileFirst runtime environments.

Upgrading server farms for MobileFirst Server V7.1.0

About this task

If you run multiple servers, and if you do not use WebSphere Application Server Network Deployment, the configuration of server farms is required since V6.2.0. The configuration of server farms has also changed in V7.0.0. In this step, you upgrade or define your server farms to be compatible with MobileFirst Server V7.1.0. Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-54. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.2.0.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.0.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.1.0 to MobileFirst Server V7.1.0.x (fix pack or interim fix)	No

Table 7-55. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are partially upgraded.

For more information, see “List of JNDI properties for MobileFirst Server administration” on page 6-101.

If you upgrade from V6.1.0 or earlier, see “Installing a server farm” on page 6-127.

If you upgrade from V6.2.0 or V6.3.0, and you already defined a farm configuration, update your farm configuration as follows:

WebSphere Application Server Liberty

Optionally, you can remove some unused JNDI properties and configure the heartbeat rate and timeout values.

- The following JNDI properties in the `server.xml` file are not used any more. You can remove them.

```
<jndiEntryjndiName="ibm.worklight.farm.type"
value="File"/>
<jndiEntryjndiName="ibm.worklight.farm.definition.location" value="<plugin xml file location>"/>
```

- You can configure the heartbeat rate and timeout values by defining the following JNDI properties in the `server.xml` file.

```
<jndiEntryjndiName="worklight.admin.farm.heartbeat"
value="<heartbeat rate in minutes>"/>
<jndiEntryjndiName="ibm.worklight.admin.farm.missed.heartbeats.timeout"
value="<number of missed heartbeats before considering the server as being down>"/>
```

Tomcat

Optionally, you can remove some unused JNDI properties and configure the heartbeat rate and timeout values.

- The following JNDI properties in the `server.xml` file are not used any more. You can remove them.

```
<Environment name="ibm.worklight.farm.type" value="File" type="java.lang.String"
override="false"/>
<Environment name="ibm.worklight.farm.definition.location"
value="<plugin xml file location>"
type="java.lang.String" override="false"/>
```

- You can configure the heartbeat rate and timeout values by defining the following JNDI properties in the `server.xml` file.

```
<Environment name="worklight.admin.farm.heartbeat" value="<heartbeat rate in minutes>"
type="java.lang.String" override="false"/>

<Environmentname="ibm.worklight.admin.farm.missed.heartbeats.timeout"
value="<number of missed heartbeats before considering the server as being down>"
type="java.lang.String"override="false"/>
```

WebSphere Application Server

Required modifications

You must set new JNDI properties in MobileFirst Administration Services applications and every MobileFirst runtime application.

1. Open the WebSphere Application Server administration console.
2. On the navigation pane, select **Applications** > **Application Types** > **Websphere enterprise applications**.
3. Select the application that you want to configure.
4. In **Web Module Properties**, click **Environment entries for Web Modules** to display the JNDI properties.
5. Set the **ibm.worklight.admin.jmx.user** property to a user name that has access to the SOAP connector.
6. Set the **ibm.worklight.admin.jmx.pwd** property to the password of the user that you declared in the **ibm.worklight.admin.jmx.user** property.
7. Verify that the **ibm.worklight.admin.jmx.connector** property is set to **SOAP**.

Optional modifications

You can configure the heartbeat rate and timeout values by defining the **worklight.admin.farm.heartbeat** and **ibm.worklight.admin.farm.missed.heartbeats.timeout** JNDI properties in the MobileFirst Administration Services application.

Restore the MobileFirst Server configuration

In this step you restore the required configurations of MobileFirst 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 MobileFirst Server and Application Center" on page 7-25.

Restore as follows:

- For the applications: The MobileFirst runtime environments.
- For the JDBC data source: The runtime and reports database.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-56. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0.x to MobileFirst Server V7.1.0	No

Table 7-56. Is this step required for your upgrade path? (continued).

Product or component versions	Required?
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

Table 7-57. System status after this step.

Module	Status
Application Center	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.
MobileFirst Server	Embedded Liberty, Liberty on Windows, and Tomcat on Windows are stopped. All other instances are upgraded.

Restart the application server

In this final step, you restart the application server.

About this task

Now that the upgrade of MobileFirst Server is completed, restart your application server.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-58. Is this step required for your upgrade path?

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0.x to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-59. System status after this step

Module	Status
Application Center	All instances are upgraded.
MobileFirst Server	All instances are 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 MobileFirst Server is upgraded. All applications that you previously deployed are available, along with their environments (those that are supported by MobileFirst Server V7.1.0).

The URL of the MobileFirst Operations Console changed. If you did not specify a context root in the Ant file, its context root is `/worklightconsole`.

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`**.

Read from the following tables whether this step is required for your upgrade path.

Table 7-60. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.0.x to MobileFirst Server V7.1.0	Yes
Worklight Server V6.1.x to MobileFirst Server V7.1.0.x	Yes
Worklight Server V6.2.x to MobileFirst Server V7.1.0	No
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	No
MobileFirst Server V7.0.0.x to MobileFirst Server V7.1.0	No
V7.1.0 to V7.1.0.x (fix pack or interim fix)	No

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 MobileFirst Platform Foundation for iOS V7.1.0.

Note: In the following code samples, *mf_install_dir* is the directory where you installed MobileFirst Server.

1. In the initialization commands of the Ant script, replace the path as follows:

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="mf_install_dir/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

   With

```
<taskdef resource="com/worklight/ant/deployers/antlib.xml">
  <classpath>
    <pathelement location="mf_install_dir/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

2. Replace <app-deployer> calls.

   ```
   <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="passw
     <deploy-app runtime="project-name" file="myApp.wlapp"/>
   </wladm>
   ```

   Set the placeholders as follows:

   - For *worklightadmin*, substitute the actual context root of the MobileFirst administration services web application
   - For *username* and *password*, pick a user that is in the role **worklightadmin** or **worklightdeployer**.

3. Replace <adapter-deployer> calls.

   ```
   <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="passw
     <deploy-adapter runtime="project-name" file="myAdapter.adapter"/>
   </wladm>
   ```

   Set the placeholders as follows:

   - For *worklightadmin*, substitute the actual context root of the MobileFirst 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 the **wladm** Ant task, see "Administering MobileFirst applications through Ant" on page 12-12.

Additional MobileFirst Server V7.1.0 upgrade information

This section contains additional information that may be of use if you have additional test or pre-production databases that must be updated, if you need to update HTTP redirections on networked servers, if you are manually upgrading the application server, or in the event of a failed upgrade.

Recovering from an unsuccessful upgrade to MobileFirst Server V7.1.0

Instructions for how to recover from a failed installation or to revert to the previous version of Worklight Server or MobileFirst Server.

About this task

If the MobileFirst Server upgrade fails for any reason, use the following procedure to restore the previous Worklight Server or MobileFirst Server version.

Procedure

The **Roll Back** button of IBM Installation Manager is not supported for MobileFirst Server. Therefore, to return to the previous version:

1. Uninstall MobileFirst Server, with IBM Installation Manager.
2. Install the old version of Worklight Server or MobileFirst 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 runtime database" on page 7-50.
4. Restore the application server. For more information, see "Back up your application server" on page 7-35.
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 MobileFirst Server administration during the upgrade

You can manually install the MobileFirst Server administration as part of the MobileFirst Server upgrade.

Since IBM Worklight Foundation V6.2.0, you must install the administration components. To manually set up the MobileFirst Server administration, follow the steps detailed in "Manually installing MobileFirst Server administration" on page 6-65.

Manually upgrading the MobileFirst Server V7.1.0 databases

Follow these instructions to manually update the MobileFirst project databases. First set up the MobileFirst Server administration environment.

Note: The reports database, referenced as `WLREPORT`, was deprecated in IBM MobileFirst Platform Foundation for iOS V7.0.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the reports database is optional in this release and previous releases. Also note that the use of the reports database is redundant with the MobileFirst Operational Analytics console in this release and previous releases, because of the deprecation of the reports database.

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 preproduction databases as part of your production environment, each served by a different runtime database or schema, you can use this procedure to update their schemas.

Procedure

Updating the reports (by default `WLREPORT`) and Application Center (by default `APPCNTR`) databases is done by running a sequence of database scripts.

Updating the runtime database (by default `WRKLGHT`) and administration (by default `WLADMIN`) databases is done by running a sequence of database scripts, if you upgrade from IBM Worklight Foundation V6.2.0 or later.

For the `WRKLGHT` database, if you upgrade from IBM Worklight V6.1.0 or earlier, you must apply the proper sql scripts to upgrade the databases to IBM MobileFirst

Platform Foundation for iOS V7.1.0. If you migrate the database from V6.1.0 to V6.2.0, you have some additional steps to perform, which are detailed in the following sections for each database.

For the WRKLGHT and WLADMIN databases, if you upgrade from V6.2.0 or later, you must apply the proper sql scripts to upgrade the databases to IBM MobileFirst Platform Foundation for iOS V7.1.0.

For the WLREPORT and APPCNTR databases: apply the proper sql scripts to upgrade the databases to IBM MobileFirst Platform Foundation for iOS V7.1.0.

Scripts for DB2

For an upgrade from IBM Worklight 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 IBM Worklight 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 IBM Worklight V6.1.0.x to IBM Worklight Foundation V6.2.0:

Important:

- First, you must install the MobileFirst Server, so that the administration database exists. Then, you must run the data migration tool, and run the following sql scripts. For more information, see the section data migration tool.
- If you manually added an index I_USERPRF_USERID to the runtime database to solve the performance problem that is described in the technote IBM Worklight queries on the GADGET_USER_PREF table might take time, you must comment the corresponding index, as described in "Commenting the I_USERPRF_USERID index from the sql scripts" on page 7-67.
- 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)

For an upgrade from IBM Worklight Foundation V6.2.0.x to IBM MobileFirst Platform Foundation for iOS V6.3.0:

- WorklightServer/databases/upgrade-worklightadmin-62-63-db2.sql (for WLADMIN)
- WorklightServer/databases/upgrade-worklight-62-63-db2.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-62-63-db2.sql (for REPORTS)

- `ApplicationCenter/databases/upgrade-appcenter-62-63-db2.sql` (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V6.3.0.x to IBM MobileFirst Platform Foundation for iOS V 7.0.0:

- `WorklightServer/databases/upgrade-worklightadmin-63-70-db2.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-63-70-db2.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-63-70-db2.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-63-70-db2.sql` (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V7.0.0.x to IBM MobileFirst Platform Foundation for iOS V7.1.0:

- `WorklightServer/databases/upgrade-worklightadmin-70-71-db2.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-70-71-db2.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-70-71-db2.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-70-71-db2.sql` (for APPCNTR)

These scripts are applied similarly to steps 4 and 6 in “Setting up your DB2 database manually” on page 11-20

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 IBM Worklight V5.0.6.x 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 IBM Worklight 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 IBM Worklight V6.1.0.x to IBM Worklight Foundation V6.2.0:

Important:

- First, you must install the MobileFirst Server, so that the administration database exists. Then, you must run the data migration tool, and run the following sql scripts. For more information, see the section data migration tool.
- If you manually added an index `I_USERPRF_USERID` to the runtime database to solve the performance problem that is described in the technote IBM Worklight queries on the GADGET_USER_PREF table might take time, you must comment the corresponding index, as described in "Commenting the I_USERPRF_USERID index from the sql scripts" on page 7-67.
- `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)

For an upgrade from IBM Worklight Foundation V6.2.0.x to IBM MobileFirst Platform Foundation for iOS V6.3.0:

- `WorklightServer/databases/upgrade-worklightadmin-62-63-mysql.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-62-63-mysql.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-62-63-mysql.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-62-63-mysql.sql` (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V6.3.0.x to IBM MobileFirst Platform Foundation for iOS V7.0.0:

- `WorklightServer/databases/upgrade-worklightadmin-63-70-mysql.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-63-70-mysql.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-63-70-mysql.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-63-70-mysql.sql` (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V7.0.0.x to IBM MobileFirst Platform Foundation for iOS V7.1.0:

- `WorklightServer/databases/upgrade-worklightadmin-70-71-mysql.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-70-71-mysql.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-70-71-mysql.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-70-71-mysql.sql` (for APPCNTR)

These scripts are applied similarly to step 1.b in "Setting up your MySQL database manually" on page 11-28.

Scripts for Oracle

Remember: To upgrade from V6.1.0 or earlier, the Oracle user must have the `CREATE VIEW` privilege.

For an upgrade from IBM Worklight V5.0.6.x 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 IBM Worklight 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 IBM Worklight V6.1.0.x to IBM Worklight Foundation V6.2.0:

Important:

- First, you must install the MobileFirst Server, so that the administration database exists. Then, you must run the data migration tool, and run the following sql scripts. For more information, see the section data migration tool.
- If you manually added an index `I_USERPRF_USERID` to the runtime database to solve the performance problem that is described in the technote IBM Worklight queries on the GADGET_USER_PREF table might take time, you must comment the corresponding index, as described in “Commenting the I_USERPRF_USERID index from the sql scripts” on page 7-67.
- `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)

For an upgrade from IBM Worklight Foundation V6.2.0.x to IBM MobileFirst Platform Foundation for iOS V6.3.0:

- `WorklightServer/databases/upgrade-worklightadmin-62-63-oracle.sql` (for WLADMIN)
- `WorklightServer/databases/upgrade-worklight-62-63-oracle.sql` (for WRKLGHT)
- `WorklightServer/databases/upgrade-worklightreports-62-63-oracle.sql` (for REPORTS)
- `ApplicationCenter/databases/upgrade-appcenter-62-63-oracle.sql` (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V6.3.0.x to IBM MobileFirst Platform Foundation for iOS V7.0.0:

- WorklightServer/databases/upgrade-worklightadmin-63-70-oracle.sql (for WLADMIN)
- WorklightServer/databases/upgrade-worklight-63-70-oracle.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-63-70-oracle.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-63-70-oracle.sql (for APPCNTR)

For an upgrade from IBM MobileFirst Platform Foundation for iOS V7.0.0.x to IBM MobileFirst Platform Foundation for iOS V7.1.0:

- WorklightServer/databases/upgrade-worklightadmin-70-71-oracle.sql (for WLADMIN)
- WorklightServer/databases/upgrade-worklight-70-71-oracle.sql (for WRKLGHT)
- WorklightServer/databases/upgrade-worklightreports-70-71-oracle.sql (for REPORTS)
- ApplicationCenter/databases/upgrade-appcenter-70-71-oracle.sql (for APPCNTR)

These scripts are applied similarly to step 3 in “Setting up your Oracle database manually” on page 11-32.

Commenting the I_USERPRF_USERID index from the sql scripts

Before you apply the sql scripts that upgrade the WRKLGHT database from the previous version 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. You might have added it manually to solve the performance problem that is described in the technote IBM Worklight queries on the GADGET_USER_PREF table might take time.

If this index exists, you must comment the statement that creates this index, in the script WorklightServer/databases/upgrade-worklight-61-62-<*dbms*>.sql.

Comment the following line that refers to its creation before running it:

```
INDEX I_USERPRF_USERID ON GADGET_USER_PREF (USER_ID);
```

The data migration tool

In IBM Worklight Foundation V6.2.0, the process of updating the runtime database (by default WRKLGHT) required to move the runtime administrative data to the administration database. When you upgrade from IBM Worklight V6.1.0 or older, before you call the script WorklightServer/databases/upgrade-worklight-61-62-<*dbms*>.sql, run the data migration tool to migrate the administration database to the new administration database. Then, run the SQL scripts from V6.2.0 to the current release 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 the production data might be corrupted or lost.

The data migration tool and the database upgrade scripts are both contained in the MobileFirst Server directory that you just installed.

Running the data migration tool

Before you run the tool in command line, make sure that the library `worklight-ant-deployer.jar` of the MobileFirst version that you installed is in your current directory, or that your `CLASSPATH` variable references the directory it is in. Example on a Unix/Linux machine:

```
# Go to the directory library that worklight-ant-deployer.jar is in
$ cd $product_install_dir/WorklightServer
# Print the usage
$ java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration.MigrationTool usage
Usage:
        java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration.MigrationTool [optio

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 with DB2 as DBMS
$ java -cp worklight-ant-deployer.jar com.ibm.worklight.config.dbmigration.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
```

To run the data migration tool, you must add the database drivers to the class path. For example, to migrate a DB2 database, you must add the `db2jcc4.jar` file and the license JAR file (for example `db2jcc_license_cu.jar`), to the class path:

```
$ java -cp worklight-ant-deployer.jar:/path/to/db2jcc4.jar:/path/to/db2jcc_license_cu.jar
com.ibm.worklight.config.dbmigration.MigrationTool usage
```

Note:

- The name of the project to create (worklight in the example) must be the context root where the MobileFirst 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.

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 V7.1.0 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 in the section for your database type.

This task requires the use of a database account that can modify the content of the runtime database. It also requires basic knowledge to run SQL scripts on the database. For DB2, you need to know how to run the DB2 command line processor or the command line program. For Oracle, make sure that you know how to run SQLPlus. This task is only needed for Oracle or DB2. It is not needed for MySQL.

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

--Value of X = (MAX(ID) OF PUSH_DEVICES - LAST_NUMBER OF PUSHDEVICE_SEQ) + 20. For exa
--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) +
--MAX(ID) FROM SCHEMANAME.PUSH_SUBSCRIPTIONS returned 100, then Y = 101 and SELECT --L
--FROM ALL_SEQUENCES (where SEQUENCE_NAME='PUSHSUBSCRIPTION_SEQ' is 50), then Y=100-50
```

```
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;
```

MySQL

No action is required.

Manually upgrading the application server

Follow these instructions to manually upgrade the application server.

The recommended way to upgrade MobileFirst 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 7-40 and "Upgrade the MobileFirst Server runtime environment" on page 7-54, use the following procedure:

- Upgrade the databases manually as specified in section "Manually upgrading the MobileFirst Server V7.1.0 databases" on page 7-62.
- Review the manual installation procedures at:
  - "Manually installing Application Center" on page 6-218
  - "Deploying a project WAR file and configuring the application server manually" on page 11-37
- Update the items manually. This includes, at a minimum:
  - The WAR file for the Application Center console, Application Center services, and the MobileFirst Operations Console.
  - The MobileFirst library `worklight-jee-library.jar`.
  - The MobileFirst 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 MobileFirst Server" on page 11-37.
- If you upgrade from IBM MobileFirst Platform Foundation for iOS V6.3.0 or earlier, and you use DB2 as a DBMS, you must update the schemas as described in "Updating DB2 schema names in the case of a manual installation" on page 7-71.

Verifying and updating the HTTP redirections for MobileFirst Server V7.1.0

If you are upgrading from IBM Worklight V6.1.0 or earlier, to MobileFirst Server on a clustered application server environment, you must also update IBM HTTP Server after you install IBM MobileFirst Platform Foundation for iOS V7.1.0.

If your MobileFirst Server upgrade is to be installed on a WebSphere Application Server Network Deployment clustered environment or a WebSphere Application Server Liberty profile farm, you might have to update IHS after you install MobileFirst Server V7.1.0. For general information about installing these types of application server, see:

- “Setting up IBM MobileFirst Platform Foundation for iOS in WebSphere Application Server cluster environment” on page 6-301
- “Setting up HTTP Server in a WebSphere Application Server Liberty profile farm” on page 6-312

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

If you are upgrading manually from IBM MobileFirst Platform Foundation for iOS V6.3.0, or earlier, to MobileFirst Server on a clustered application server environment, and if you are using DB2 for the runtime database or the reports database, you must update the DB2 schema.

About this task

Since IBM MobileFirst Platform Foundation for iOS V7.0.0, for the runtime database and the reports database, IBM MobileFirst Platform Foundation for iOS 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. Optional: If there are double quotation marks around the value of the currentSchema attribute, you must remove them. For example, change currentSchema='"wrkschem"' or currentSchema="&quot;wrkschem&quot;" 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;'`.

Upgrading with the Server Configuration Tool

Upgrade an older configuration that was deployed with the Server Configuration Tool.

Before you begin

Before you start to upgrade, you must complete the following steps:

- Prepare the passwords of the databases that are to be upgraded, and the passwords for the WebSphere Application Server, if this is the application server that you use in your configuration. For more information, see “Gathering information for MobileFirst Server V7.1.0 upgrades” on page 7-17.
- Shut down your application server. This is mandatory if you use Apache Tomcat, or Liberty stand-alone servers. For more information, see “Shutting down the application server to be upgraded” on page 7-48.

To be able to restore your previous deployment if the upgrade fails, you must complete the following steps:

- Back up your administration and runtime databases. For more information, see “Back up the runtime database” on page 7-50.
- Find your previous project WAR file, or project WAR files, which are already deployed. You can then redeploy them. For more information, see “Identify the MobileFirst WAR file and prepare the Ant deployment script” on page 7-21.
- Back up your application server configuration. For more information, see “Back up your application server” on page 7-35.

About this task

Since IBM MobileFirst Platform Foundation for iOS V7.1.0, you can upgrade an older configuration that is already deployed, by using the Server Configuration Tool. You can perform this upgrade with the Server Configuration Tool only on a configuration that uses the architecture of MobileFirst Server that was introduced in V6.2.0. You can then upgrade from V6.2.0 or later.

Read from the following tables whether this step is required for your upgrade path and what the system status is afterward if both Application Center and MobileFirst Server are on the same application.

Table 7-61. Is this step required for your upgrade path?.

Product or component versions	Required?
Worklight Server V5.0.6.x to MobileFirst Server V7.1.0	No
Worklight Server V6.0.x to MobileFirst Server V7.1.0	No
Worklight Server V6.1.x to MobileFirst Server V7.1.0	No
Worklight Server V6.2.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V6.3.x to MobileFirst Server V7.1.0	Yes
MobileFirst Server V7.0.0 to MobileFirst Server V7.1.0	Yes
V7.1.0 to V7.1.0.x (fix pack or interim fix)	Yes

Table 7-62. System status after this step.

Module	Status
Application Center	STOPPED (embedded Liberty, Liberty on Windows, Tomcat on Windows), UPGRADED (other cases)
MobileFirst Server	STOPPED (all instances)

Limitations

- This procedure concerns only the MobileFirst Server upgrade, not the Application Center upgrade.
- You can perform an upgrade with the Server Configuration Tool only if the configuration was deployed with the Server Configuration Tool from IBM Worklight Foundation V6.2.0, or later.
- You cannot perform an upgrade with the Server Configuration Tool on a configuration that was deployed manually, or with Ant tasks.
- You cannot perform an upgrade with the Server Configuration Tool on a server farm installation.

Procedure

To perform this upgrade, a new button **Upgrade a previous configuration** is defined in the Server Configuration Tool. This button is only available when you select a deployed configuration.

1. Select the previously deployed configuration.
2. Start the upgrade by right-clicking the configuration in the navigation tree, then selecting **Configurations** > **Upgrade a previous Configuration**.
3. A modal dialog opens.
   a. Select the configuration to upgrade.
   b. If you decide to upgrade the reports database for each runtime that is deployed, select the **Upgrade the Runtime REPORT database** check-box. By default, the upgrade of the reports database is not activated.
   c. If your installation is on WebSphere Application Server, and the security of the server is activated, you must enter the administration password of the application server.

   d. Enter the passwords of the users that have the privileges to upgrade the administration and runtime databases.
   e. Enter or confirm the WAR file path for each runtime. The WAR file must have the same base name as the WAR file that was initially deployed.
   f. Click **OK**.
4. The upgrade procedure begins. The procedure includes a database upgrade. If the database contains a large number of entries, this step might require one hour or more to complete.

What to do next

If your upgrade fails, you can complete the following steps to solve the issue, as the execution of the upgrade is reentrant:

1. Analyze the issues that you can find in the log file, or directly in the console.
2. Correct the issues that you found. If the Server Configuration Tool was interrupted in the middle of a database upgrade script, you must complete that upgrade script manually. The upgrade scripts can be found in `*product_install_dir*/WorklightServer/databases`.
3. Start the upgrade again. The upgrade can be run twice on the same configuration.

If the upgrade fails definitively, you must restore your previous deployment with the databases and application server back-ups that you did before starting the upgrade procedure (see section *Before you begin*).

Rolling upgrade procedure to apply a fix pack to IBM MobileFirst Platform Foundation for iOS V7.1.0 in stateful mode

If your installation of MobileFirst Server V7.1.0 is configured in stateful mode, meaning that it uses sticky sessions with an attribute store of type `HTTPSESSION`, you can perform a rolling upgrade to apply a fix pack or an interim fix to an installation of MobileFirst Server V7.1.0, without downtime of the MobileFirst runtime environment. Performing this rolling upgrade ensures that there is no interruption of service for the mobile applications that query the MobileFirst Server.

Note: Since MobileFirst Server V7.1.0, new MobileFirst runtime environments are configured in stateless mode by default (distributed attribute store, or type `DATABASE` or `EXTREMESCALE`). The following procedure, which runs both a previous, and a more recent installation of MobileFirst in parallel for some time, must not be used for a stateless installation. In a stateless installation, HTTP requests of a same session can be sent indifferently to the previous, or the more recent installation. If there is a different behavior in both installations, this can prevent mobile applications from working correctly.

To perform this rolling upgrade, you must install the upgraded version of IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS 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 “Upgrading to IBM MobileFirst Platform Foundation for iOS V7.1.0” on page 7-1.

Important: This procedure applies to a MobileFirst Server installation, including the Administration Services, the MobileFirst Operations Console, and the MobileFirst 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 MobileFirst Platform Foundation for iOS, 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 MobileFirst Platform Foundation for iOS without downtime.

You must review, adapt, and test this upgrade procedure for your production environment. Other components that interact with IBM MobileFirst Platform Foundation for iOS 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 a previous installation of MobileFirst Server to an upgraded installation of MobileFirst Server. This is done without losing any data in the former databases, and without visible impact for users of 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 MobileFirst 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 MobileFirst push notification architectures” on page 8-170.

Verify that your installation of MobileFirst Server V7.1.0 is configured in stateful mode (attribute store of the type HTTPSESSION).

Since MobileFirst Server V7.1.0, new MobileFirst runtime environments are configured in stateless mode by default (distributed attribute store, or type `DATABASE` or `EXTREMESCALE`). The following rolling upgrade procedure, which runs both a previous, and a more recent installation of MobileFirst in parallel for some time, must not be used for a stateless installation. In a stateless installation, HTTP requests of a same session can be sent indifferently to the previous, or the more recent installation. If there is a different behavior in both installations, this can prevent mobile applications from working correctly.

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 MobileFirst application or adapter is uploaded to the MobileFirst 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 MobileFirst application is uploaded to the MobileFirst Server and the runtimes do not all have the same version, it might trigger arbitrary direct update sessions for the users of MobileFirst apps, depending on the server to which they were routed. Identify all MobileFirst 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 instances of WebSphere Application Server 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 MobileFirst Platform Foundation for iOS V7.1.0.x in that duplicated environment, for example in a new cluster. You must use the existing administration database, MobileFirst runtime database, and MobileFirst 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Server is confirmed to work correctly, shut down the old servers. If required, uninstall the old IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS, assuming the following topology: IBM MobileFirst Platform Foundation for iOS 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 your current installation of the product 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 artifact until the upgrade procedure is complete.

Installing the IBM MobileFirst Platform Foundation for iOS fix pack in a new cluster

You must create a new cluster, in which you install the IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS fix pack in this cluster. You can install this fix pack with the IBM MobileFirst Platform Foundation for iOS Ant tasks, or manually.
   - To install this fix pack with the IBM MobileFirst Platform Foundation for iOS Ant tasks, follow the steps 5 - 9 on page 7-78.
   - To install this fix pack manually, follow the steps 10 on page 7-79 - 14 on page 7-79.

Installing the fix pack with the IBM MobileFirst Platform Foundation for iOS Ant tasks:

5. With IBM Installation Manager, install the IBM MobileFirst Platform Foundation for iOS 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 product 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 MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Server administration components.
8. Run the `adminstall` target of the Ant file.
9. Run the `install` target of the Ant file that installs the MobileFirst runtime environment. If you have more than one MobileFirst 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 IBM MobileFirst Platform Foundation -->
  <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 MobileFirst runtime databases in the cluster.
    - For instructions about the administration database, see the following documentation:
      - "Configuring WebSphere Application Server for DB2 manually for MobileFirst Server administration" on page 6-67
      - "Configuring WebSphere Application Server for Oracle manually for the MobileFirst Server administration" on page 6-79
    - For instructions about the runtime database, see the following documentation:
      - "Configuring WebSphere Application Server for DB2 manually" on page 11-22
      - "Configuring WebSphere Application Server for Oracle manually" on page 11-34
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 MobileFirst Platform Foundation for iOS 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 MobileFirst Server administration manually" on page 6-84, 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 MobileFirst runtime environment as documented in "Configuring WebSphere Application Server for MobileFirst Server administration manually" on page 6-84, with the following change:

    In **Environment entries for Web modules** for the MobileFirst 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 MobileFirst Platform Foundation for iOS

You must complete the configuration of the new installation of IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Operations Console application. For more information, see "Configuring MobileFirst Server" on page 6-136.

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 MobileFirst Operations Console.
   - For each MobileFirst runtime environment.

Verifying the new installation of IBM MobileFirst Platform Foundation for iOS

You must make sure that IBM MobileFirst Platform Foundation for iOS is installed properly.

Procedure

1. Start all the servers in the **`Worklight61FP1`** cluster. This action starts all the MobileFirst 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 MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS

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 MobileFirst Platform Foundation for iOS in WebSphere Application Server cluster environment" on page 6-301.

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 MobileFirst Platform Foundation for iOS.

Uninstalling IBM MobileFirst Platform Foundation for iOS from the old cluster

You must uninstall IBM MobileFirst Platform Foundation for iOS 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 MobileFirst artifacts because the upgrade procedure is complete.
3. Uninstall IBM MobileFirst Platform Foundation for iOS 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.

Upgrading, or applying a fix pack to the MobileFirst Data Proxy

Follow these steps to upgrade the MobileFirst Data Proxy and the Trust Association Interceptor (TAI).

Before you begin

You must run IBM Installation Manager to upgrade the installation directory of IBM MobileFirst Platform Foundation for iOS. For more information, see "Running IBM Installation Manager and completing the Application Center upgrade" on page 7-40.

The following procedure installs the upgraded versions of the MobileFirst Data Proxy WAR file and the TAI (Trust Association Interceptor) on your disk.

Note: If you installed Application Center, you might want to upgrade or apply a fix pack only to the MobileFirst Data Proxy, and not to the Application Center. To do this, you can run IBM Installation Manager to install IBM MobileFirst Platform Foundation for iOS in a different location, without installing Application Center:

- For a graphic installation, in the window **Configuration Choice**, select **No** to the question **Install the IBM MobileFirst Platform Application Center**.
- For a command-line installation, use the silent installation file that is named `install-no-appcenter.xml`, which is described in "Working with sample response files for IBM Installation Manager" on page 6-35.

About this task

If you installed the MobileFirst Data Proxy with Ant tasks, follow the steps 1 to 6.

If you installed the MobileFirst Data Proxy manually, follow the steps 7 to 11.

Procedure

For an installation with Ant tasks:

1. Locate the Ant file that you used to perform the installation.
2. Verify that the **`taskdef`** for the `worklight-ant-deployer.jar` uses the correct directory, which contains the upgraded installation of MobileFirst Server V7.1.0.

   Review the following example. You must 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` file in the **`taskdef`** tag:

   ```
   <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 must not be overlooked because it defines the version of IBM MobileFirst Platform Foundation for iOS that you use to deploy the WAR file and update the TAI.
3. Stop your application server.
4. Run the following command:

   `product_install_dir``/shortcuts/ant -f <`*`your file`*`> minimal-update`

This command has the following effects:

- It updates the WAR file of the MobileFirst Data Proxy web app.
- On WebSphere Application Server Liberty profile, and WebSphere Application Server stand-alone server, it updates the TAI.

5. On WebSphere Application Server Network Deployment, replace the TAI JAR file by the new JAR file, which is in *product_install_dir*/WorklightServer/external-server-libraries, on all nodes of your WebSphere Application Server cell.

   For more information, see the instructions to install the MobileFirst Data Proxy manually.
6. Restart the application server.

For a manual installation:

7. Replace the MobileFirst Data Proxy WAR file in your application server.

   You find the WAR file in *product_install_dir*/Datastore/imf-data-proxy.war.
8. Stop the application server.
9. Replace the TAI JAR file in your application server.

   You find the JAR file in *product_install_dir*/WorklightServer/external-server-libraries/com.ibm.worklight.oauth.tai_*.jar.
10. On WebSphere Application Server Liberty profile, replace the TAI feature manifest in your application server.

    You find the manifest in *product_install_dir*/WorklightServer/external-server-libraries/OAuthTai-1.0.mf.
11. Restart the application server.

Applying a fix pack to IBM MobileFirst Platform Operational Analytics

Learn how to use Ant tasks to upgrade MobileFirst Operational Analytics V7.1.0 to a fix pack or interim fix, if you installed this component with Ant tasks.

Before you begin

On WebSphere Application Server, applying a fix pack creates a significant peak of memory usage in the heap. If WebSphere Application Server has insufficient memory heap, this might cause OutOfMemory issues, and seriously impact the performance of the application server. Before you perform the following procedure, verify that your server on WebSphere Application Server has the memory settings that are described at “Setting up a production cluster” on page 13-85.

About this task

Use this procedure if you installed MobileFirst Operational Analytics with Ant tasks, as documented in “Installing MobileFirst Operational Analytics with Ant tasks” on page 6-191.

If you installed MobileFirst Operational Analytics manually, you must perform a manual update of the WAR file, or EAR file. For more information, see “Installing the IBM MobileFirst Platform Operational Analytics” on page 6-191.

Procedure

For an upgrade from MobileFirst Server V7.1.0 to a fix pack or an interim fix

1. Find the Ant file that you created in “Sample configuration files for MobileFirst Operational Analytics” on page 15-71..
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 MobileFirst Server V7.1.0.
3. Run the `minimal-update` target of the Ant file on the system where you installed MobileFirst Operational Analytics:

   ```
   product_install_dir/shortcuts/ant -f your_file minimal-update
   ```
4. Restart the server.

What to do next

If you installed more than one MobileFirst Operational Analytics servers that are connected to your MobileFirst runtime environment, then you must repeat this procedure on each server where MobileFirst Operational Analytics is installed.

Applying a fix pack to Application Center installed with Ant tasks

If you installed Application Center with Ant tasks, learn how to use Ant tasks to upgrade Application Center V7.1.0 to a fix pack or interim fix.

Before you begin

You must run IBM Installation Manager to upgrade the installation directory of IBM MobileFirst Platform Foundation for iOS, without installing Application Center.

During the installation process with IBM Installation Manager, in the window **Configuration Choice**, select **No** to the question *Install the IBM MobileFirst Platform Application Center*.

For more information, see “Running IBM Installation Manager and completing the Application Center upgrade” on page 7-40.

With the following procedure, you install the upgraded versions of the Application Center Console and Services web applications on your disk.

About this task

Use this procedure if you installed Application Center with Ant tasks, as documented in “Installing the Application Center with Ant tasks” on page 6-216.

If you installed Application Center manually, you must perform a manual update of the WAR file, or EAR file. For more information, see “Manually installing Application Center” on page 6-218.

Procedure

For an upgrade from Application Center V7.1.0 to a fix pack or an interim fix:

1. Find the Ant file that you created in “Deploying the Application Center console and services with Ant tasks” on page 6-217.
2. Run the **`minimal-update`** target of the Ant file, on the system where you installed Application Center:

   ```
   product_install_dir/shortcuts/ant -f your_file minimal-update
   ```

3. Restart the server.

Developing MobileFirst applications

You use IBM MobileFirst Platform Command Line Interface for iOS, the MobileFirst client, and the server-side API to develop iOS applications.

This information is designed to help users develop applications for various channels by using IBM MobileFirst Platform Foundation for iOS. It is intended for developers who are familiar with iOS application development.

This section covers client-side development and server-side development topics, such as the integration with back-end services, and push notifications.

Development framework features

IBM MobileFirst Platform Foundation for iOS 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.

MobileFirst projects

Develop mobile applications within projects, build your applications, and create skins for specific devices.

MobileFirst projects

To develop your mobile applications, you must first create a project.

In your project, when you create an application, you have a main application folder, in which you can find several subfolders and files:

- One folder for the iOS environment in the application, and where you store the Objective-C code that is specific to this environment.
- 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 “Application descriptor of native API applications” on page 8-7.
- A `build-settings.xml` file, for you to prepare minification and concatenation configurations for each environment.

MobileFirst CLI overview

To help developers get a better tools experience, IBM MobileFirst Platform provides a command-line interface (CLI) tool to easily create and manage apps. The CLI enables developers to use their preferred text editors or alternative IDEs to create mobile applications.

The commands support tasks such as creating, adding, and configuring with the MobileFirst API library, adding the client-side MobileFirst properties file and performing the build and deploy of the MobileFirst application. From the command-line, you can create and deploy adapters, and test them locally. You can administer your MobileFirst project from CLI or REST services, or the Console, where you can easily control the local server and observe the logs.

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 CLI” on page 6-2.

CLI commands usage

The CLI commands are intended for IT developers to create MobileFirst projects separately from the Eclipse MobileFirst Studio.

Use the command-line interface (CLI) keywords and options from a command line window. To run the commands, you can use either **mfp** or **mobilefirst**.

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: **$ mfp add adapter MyAdapter --type http**.
- The interactive method: You enter the command with no arguments and press Enter. For example: **$ mfp add adapter**. Then, you are prompted to set the available parameters one by one.

Examples:

```
$ mfp create MyProject --app myApp
$ cd MyProject
$ mfp add api MyiOS
$ mfp add adapter MySQLAdapter --type sql
$ cd MySQLAdapter
$ mfp push localDevServer
```

Global options

You can use the following options on any MobileFirst Platform Command Line Interface for iOS command:

Option	Description
-v, --version	Prints this utility's version
-d, --debug	Produces debug log output
-dd, --ddebug	Produces verbose debug log output

Option	Description
`--no-color`	Suppresses use of special text colors for all command line output. When specified, the color settings for your operating system's console are used for all error messages, status messages and other CLI prompts.

For a complete list of the CLI commands with descriptions of their functions, see "CLI Commands" on page 15-1.

Getting started with CLI

To get started with the CLI, configure your global commands, create one MobileFirst Server definition, and create your MobileFirst project.

Procedure

1. Configure global commands. Run **mfp config**. For more information about the **mfp config** command, see "**config**" on page 15-4.
2. Create at least one MobileFirst Server definition.
   - For a local Development Server, run **mfp server create**. For more information, see "**server create**" on page 15-11.
   - For a remote server, run **mfp server add**. For more information, see "**server add**" on page 15-11.
3. Create a MobileFirst project to manage your back-end assets. Run **mfp create**. For more information, see "**create**" on page 15-6.
4. Change directories into your project. Run **cd YourProject**.
5. If you are using the local development server, start the MobileFirst Server. For the local development server, run **mfp start**. This should take between 1 and 5 minutes. For the remote server, verify with the administrator that the server is running. For more information, see "**start**" on page 15-13.
6. Verify server access by opening the Console. Run **mfp console**. For more information, see "**console**" on page 15-6.
7. Push your app to the MobileFirst Server. Run **mfp push**.

Creating an app back-end runtime

To create an application back-end runtime, use the **mfp create** command.

Procedure

1. To create your MobileFirst back-end project, run the following command:

   ```
   mfp create myProject
   ```

   where *myProject* is the name of your project. For more information, see "**create**" on page 15-6.
2. Enter **cd** *myProject* to change into your MobileFirst back-end project directory.
3. Create at least one MobileFirst Server definition.
   - For a local MobileFirst Development Server:
     - If this is the first time that you are creating a project or the existing server is stopped, run the **mfp start** command. For more information, see "**start**" on page 15-13.
     - If an existing server is already running, run the **mfp restart** command. For more information, see "**restart**" on page 15-10.

Tip: To find out if a server is running, run the **`mfp status`** command. For more information, see "**`status`**" on page 15-14.

- For a remote server, run **`mfp server add`**. For more information, see "**`server add`**" on page 15-11.

4. Deploy the back-end runtime.
   - For a local MobileFirst Development Server, run **`mfp push`**. For more information, see "**`push`**" on page 15-9.
   - For a remote server, follow the instructions to build and deploy the back-end runtime WAR file. For more information, see "Deploying the project WAR file" on page 11-5.
5. Verify that the back-end runtime has been successfully deployed by obtaining the list of runtimes that are running on the MobileFirst Server. To obtain this list, run the following command:

   ```
   mfp server info MyServerDefinition
   ```

   where *`MyServerDefinition`* is the name of the server definition that represents the target server. For example, run

   ```
   mfp server info local
   ```

   to obtain the list of runtimes on the local MobileFirst Development Server.

Results

The application back-end runtime that is deployed on the MobileFirst Server starts. You can now deploy your apps and adapters to this back-end runtime.

Troubleshooting a Request Timeout error

You can often resolve a Request Timeout error that you receive when you use the IBM MobileFirst Platform Command Line Interface for iOS by increasing the default timeout value.

Symptom

You receive a Request Timeout error while you are using the IBM MobileFirst Platform Command Line Interface for iOS.

Cause

This error occurs when the IBM MobileFirst Platform Command Line Interface for iOS does not receive a response from the IBM MobileFirst Platform Server (MobileFirst Server) within 30 seconds. This communication error is caused by one of the following situations:

- The MobileFirst Server is experiencing a high traffic load.
- There is network latency between your development system and the MobileFirst Server.
- Your MobileFirst Server failed.

Resolving the problem

To confirm that the problem is being caused by basic network latency, run a **`ping`** command to the MobileFirst Server. If the response time is longer than expected, it is a latency issue. Contact your network administrator.

Increasing the default timeout value on the system that is running the IBM MobileFirst Platform Command Line Interface for iOS can resolve this problem. The built-in default value is 30000 milliseconds (30 seconds). Increasing the value allows more time for the communication to take place, so it resolves some latency issues. To override the built-in default timeout value, you can set or increase the value of an environment variable, MFP_HTTP_DEFAULT_TIMEOUT. To set or modify this value, complete the following steps:

1. Determine if your system already has an environment variable, MFP_HTTP_DEFAULT_TIMEOUT.
   - If this variable already exists, increase the value to higher than the built-in default value of 30000.
   - If this variable does not already exist, create the variable and set it to higher than the built-in default value of 30000.
2. Rerun the failing MobileFirst Platform Command Line Interface for iOS command after you increase this environment variable.

If you still receive the Request Timeout error after you increase the timeout value to 90 seconds or longer and the ping latency is low, contact your MobileFirst Server administrator to confirm that the MobileFirst Server is running correctly.

Developing applications

Whatever the environment, the process for developing native applications shares some common elements.

Each native API application contains the following elements:

- a MobileFirst Native API project
- a project in an IDE that is appropriate to your target environment
- the MobileFirst native library for your target environment
- a client property file
- an application descriptor file

You develop according to the guidelines for iOS applications.

When you create a MobileFirst project of type Native API, various files are generated. After you create the MobileFirst native API project, you copy some of the generated files to your native development environment (IDE). If you use CocoaPods to obtain the IBM MobileFirst Platform Foundation for iOS SDK, you need to copy only a single file.

Application files

Native applications contain the following files:

- The application descriptor file: This file is the `application-descriptor.xml` file in the application root directory.
- The MobileFirst native library and the client property file: Their name and format depend on the environment.

  iOS

  - The `WorklightAPI` folder defines the MobileFirst native library.
  - The `worklight.plist` file is the client property file.

Using package management tools to obtain the SDK

If you are developing a native iOS application, you can download the IBM MobileFirst Platform Foundation iOS SDK with CocoaPods. For more information, see "Adding the IBM MobileFirst Platform Foundation iOS SDK to a new application with CocoaPods" on page 8-13.

Complementary project for native applications

Native iOS applications need another project, which you create in the Xcode IDE. You can develop in Objective-C or Swift.

Build and deployment

You build and deploy native API applications by creating a `.wlapp` file and uploading it to the MobileFirst Operations Console. For more information about deployment, see "Deploying applications and adapters to MobileFirst Server" on page 11-84.

Integrating IBM MobileFirst Platform Foundation for iOS APIs into native applications

To develop a native application, you must create a MobileFirst project of type Native API. The content of the MobileFirst project varies, depending on the target mobile environment. Then, you transfer files that are generated by the process of creating this project to the IDE for your target environment. You use the API that corresponds to your target environment.

APIs for target environments

For iOS, use the MobileFirst Objective-C client-side API. Specific steps apply if you use Apple Swift, a language that is compatible with Objective-C. For more information, see "Developing native applications" on page 8-7.

Creating a native project with the MobileFirst Platform Command Line Interface for iOS

The process to integrate IBM MobileFirst Platform Foundation for iOS APIs into your application differs depending on whether the MobileFirst project exists or not. If you already have a MobileFirst project, you can create and add your native API application to it.

- If you do not have a MobileFirst project, you can choose from two alternatives: you can use the **`mfp push`** command, or you can use the **`mfp add api`** and **`mfp create`** commands to create a MobileFirst project, then copy some files to a project in your IDE.

  Creating the native project in your native IDE with the `mfp push` command

  1. Open your native IDE. (For example, Xcode for iOS, or Android Studio for Android, or Visual Studio for Windows environments.)
  2. Create your native application project in the native IDE.
  3. Navigate, using the command line, to the directory that contains your native project.
  4. Run the **`mfp push`** command. For more information see "**`push`**" on page 15-9.

  Creating the native project with the `mfp add api` and `mfp create` commands

1. Run the **mfp create** *MyProject* command, where *MyProject* is the name of your project. For more information about the **create** command, see "**create**" on page 15-6.
2. Change directories to the directory that was created for your project: **cd** *MyProject*.
3. Run the command **mfp add api***MyProject* **--environment** *environment*, where *environment* is the target environment. For more information about the **add api** command, see "**add api**" on page 15-3.

- If you already have a MobileFirst project, you can create and add your native API application to it. In the directory that contains your project:
  - Run the **add api** command. For more information, see "**add api**" on page 15-3.

Developing native applications

After you create the native API application, you edit the application descriptor and client property files, then set up your Xcode development environment for working with MobileFirst projects. If you want to work with Apple Swift language, you create a Swift project.

Note: The **Keychain Sharing** capability is mandatory while running iOS apps in the iOS Simulator when using Xcode 8. You need to enable this capability manually before building the Xcode project.

Application descriptor of native API applications

In the application descriptor, you define various aspects of your native iOS application.

The application descriptor file is a metadata file in which you 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="7.1.0"
  version="1.0"
  securityTest="security test name"
  bundleId="com.ios"
  xmlns="http://www.worklight.com/native-ios-descriptor"
  applicationId="ios">
  <displayName>application display name</displayName>
  <description>application description</description>
  <accessTokenExpiration>3600</accessTokenExpiration>
  <userIdentityRealms>Realm1, Realm2</userIdentityRealms>
  <licenseAppType>APPLICATION</licenseAppType>
  <pushSender password="${push.apns.senderpassword}"/>

  <!--
  Define the target category for generating ADDRESSABLE DEVICE license reports
  Possible values are 'UNDEFINED', 'B2E', 'B2C'
  -->
  <targetCategory>B2E</targetCategory>
</nativeIOSApp>
```

The `<nativeIOSApp>` element is the root element of the descriptor.

id This attribute specifies the ID of the application. The ID must be identical to the application folder name and the value of **applicationId**. 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 MobileFirst Platform Foundation for iOS 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 optional attribute specifies a security configuration that is defined in the `authenticationConfig.xml` file. When a client attempts to access a protected resource, IBM MobileFirst Platform Foundation for iOS checks whether the client is already authenticated according to the security test. If the client is not yet authenticated, IBM MobileFirst Platform Foundation for iOS starts the process to obtain the client credentials and to verify them.

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

applicationId
This attribute specifies the name of the application, as specified in the `worklight.plist` file. The values in the `application-descriptor.xml` and `worklight.plist` must match, and the value of **applicationId** must be the same as the value of **id**. This attribute is required if you use the IBM MobileFirst Platform Foundation for iOS classic security model. For more information, see "MobileFirst application authenticity overview" on page 8-221.

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

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

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

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

<userIdentityRealms>
This element is a comma-separated ordered list of user identity realms for OAuth authentication. The realms should be ordered by preference. The first successfully authenticated realm in this list is selected as the user identity realm. If the list is empty, or no realm in the list was authenticated, the ID token contains no identity information. This element is optional and the default value is an empty list:

```
<userIdentityRealms>WASLTPARealm, CustomAuthenticatorRealm</userIdentityRealms>
```

Note: This attribute is used to set user identity in the OAuth-based flows. For the classic (pre-V7.0) authentication flows, see the documentation for the customSecurityTest security test.

<accessTokenExpiration>
This element is optional and defines the expiration period (in seconds) of

OAuth access tokens. The default is 3600 seconds (one hour). Client applications can use the token to access protected resources as long as the token has not expired. When the token expires, the client application has to obtain a new access token to access a protected resource. Obtaining a new access token may happen automatically without user intervention if all the realms by which the resource is protected have not expired. However, if one of the realms that is included in the security test of the resource has expired, and the realm requires user input (such as a form-based authenticator), the user has to re-authenticate.

```
<accessTokenExpiration>3600</accessTokenExpiration>
```

licenseAppType

Declares the application type for token licensing. This element is ignored if the MobileFirst Server does not have token licensing activated.

Possible values (values are case-sensitive; must be all uppercase):

- `NON_PRODUCTION`: Use this while you are developing and testing the application on the production server.

  Important: Using `NON_PRODUCTION` for a production app is a breach of the license terms.
- `APPLICATION`: Use this for most applications when you are ready to deploy the app on a production server. This is the default. The production server must have token licensing activated, and enough tokens must be available.
- `ADDITIONAL_BRAND_DEPLOYMENT`: Use this when you are ready to deploy an additional brand deployment app on a production server. The app must be an additional brand deployment of an existing app for which an `APPLICATION` token has already been used. Your organization must have purchased an Additional Brand Deployment license for you to use this option. In addition, the code in the similar application must meet specific requirements such as the percentage of code that is different. Refer to your IBM MobileFirst Platform Foundation for iOS - Additional Brand Deployment software license for details.

  The production server must have token licensing activated, and enough tokens must be available.

If token licensing is activated on the server and this element is not present in the application descriptor, then the default setting `APPLICATION` is used.

For more information, see “Licensing in MobileFirst Server” on page 2-10 and your IBM MobileFirst Platform Foundation for iOS software license.

```
<licenseAppType>APPLICATION</licenseAppType>
```

<pushSender>

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

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

<targetCategory>

Since IBM MobileFirst Platform Foundation for iOS V7.1.0, you can declare the target category of your application. With this capability, you can generate more accurate license reports for the `ADDRESSABLE DEVICE` license metric. In the `application-descriptor.xml` file, the `<targetCategory>` element is added automatically when the application is created. By default, the target category is set to `UNDEFINED`. If your application is licensed under the `ADDRESSABLE DEVICE` metric, you must edit the `<targetCategory>` element to specify the relevant target category. The possible values are B2E for IBM MobileFirst Platform Foundation Enterprise, and B2C for IBM MobileFirst Platform Foundation Consumer.

```
<!--
Define the target category for generating ADDRESSABLE DEVICE license reports
Possible values are 'UNDEFINED', 'B2E', 'B2C'
-->
<targetCategory>B2E</targetCategory>
```

</nativeIOSApp>

This tag closes the content of the application descriptor file.

```
</nativeIOSApp>
```

Client property file

This file defines the client-side properties so that your native app uses the MobileFirst native API for iOS.

The worklight.plist client property file contains the necessary information for initializing WLClient instances. Before you use this file in your native application for iOS, you must define the properties as specified in the following table.

Table 8-1. Properties of the worklight.plist file

Property	Description	Example values
protocol	The communication protocol with MobileFirst Server.	http or https
host	The host name of MobileFirst Server.	localhost
port	The port of MobileFirst Server. If you leave this value 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 MobileFirst Platform Foundation Developer Edition, you must set the value of this property to the name of your MobileFirst 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 MobileFirst environment. The value of this property must be iOSnative. **Important:** You must not modify the value of this property value.	iOSnative

Table 8-1. Properties of the `worklight.plist` file (continued)

Property	Description	Example values
`languagePreferences`	This property defines a comma-separated list of preferred languages to be used byIBM MobileFirst Platform Foundation for iOS to display system messages. This property is optional.	`en, fr, de, es`
`platformVersion`	This property defines the version number of the IBM MobileFirst Platform Foundation for iOS.	`6.3.0.00.20140813-0730`
`TRUSTEER_AUTO_INIT`	When set to `false`, this property disables the default initialization of the Trusteer Mobile SDK for MobileFirst projects with Trusteer integration. For more information, see "Integrating IBM Trusteer for iOS" on page 14-10.	`false`
`wlUid`	This property is for internal usage. You must not modify the value.	`wY/mbnwKTDDYQUvuQCdSgg==`

Methods of setting up your environment

You can prepare your environment for developing MobileFirst applications in either of two ways: by copying several files or by obtaining the files from CocoaPods.

You can get the IBM MobileFirst Platform Foundation for iOS SDKframework and library files by copying them from your MobileFirst project that you created with the MobileFirst Platform Command Line Interface for iOS. Or, you can get the framework and library files by installing the MobileFirst iOS SDK from the CocoaPods dependency manager.

Copying SDK and configuration files from the project:

You can get framework and library files that are required for MobileFirst development by creating a MobileFirst project and copying certain generated files into your Xcode project. Then, in Xcode, you configure what frameworks, libraries, and build settings to include.

Before you begin

You must have already created a MobileFirst native API project with the MobileFirst Platform Command Line Interface for iOS. For more information, see the "**`create`**" on page 15-6 command.

About this task

Make sure the MobileFirst project files are accessible from your native iOS project in the Xcode development environment. Then, by using Xcode, you specify what frameworks, libraries, and build settings to include.

Procedure

Setting up the Xcode project to include the MobileFirst API.

1. In your Xcode project add the MobileFirst files to your project:.
   a. Select the project root icon in the project explorer.
   b. From the **File** menu choose the **AddFiles** option and add the `WorklightAPI` folder and the `worklight.plist` file from your MobileFirst project.
   c. Select **Copy items if needed** and **Create groups for any added folders** options.
   d. Click **Add**.

   Note: These steps add the MobileFirst frameworks to the **Link Binary with Libraries** list in the **Build Phases** tab.
2. Make sure the following resources are linked to your project in the **Link Binary With Libraries** section of the **Build Phases** tab.
   - `IBMMobileFirstPlatformFoundation.framework`
   - `SystemConfiguration.framework`
   - `MobileCoreServices.framework`
   - `CoreData.framework`
   - `CoreLocation.framework`
   - `Security.framework`
   - `sqlcipher.framework`

     Note: Some frameworks may already be linked.
   - `libstdc++.6.dylib`
   - `libz.dylib`
   - `libc++.dylib`

   Important: If you are using Xcode 7, link `libstdc++.6.tbd`, `libz.tbd`, and `libc++.tbd`, instead of the corresponding `.dylib` files. Using Xcode 7 and iOS9 require the latest interim fix.

   Important: If you are using the latest interim fix link these framework:
   - `openssl.framework`
   - If you want to enable the JSONStore feature (which is optional in the latest interim fix) link the following frameworks:
     - `IBMMobileFirstPlatformFoundationJSONStore.framework`
     - `SQLCipher.framework`
3. If you are using the JSONStore feature, import the `IBMMobileFirstPlatformFoundationJSONStore` framework.

   For Objective C

   ```
   #import <IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.h>
   ```

   For Swift

   ```
   import IBMMobileFirstPlatformFoundationJSONStore
   ```
4. Optional: If the required frameworks do not appear in the **Build Phases** tab:.
   a. Open the **Build Settings** page.
   b. Find the **Search Paths** section.

c. Add the following entry to the **Framework Search Paths**: `$(PROJECT_DIR)/WorklightAPI/Frameworks`. If you did not copy the folder into the Xcode project, enter the full path of the `WorklightAPI` folder.

5. in the **Other Linker Flags** field, enter the following value: `-ObjC`
6. If you are using Swift see “Configuring a Swift application” on page 8-15
7. In the **Deployment** section, select a value for the **iOS Deployment Target** field that is greater than or equal to 6.0.
8. 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 8-20.

Adding the IBM MobileFirst Platform Foundation iOS SDK to a new application with CocoaPods:

You can prepare your environment for developing a IBM MobileFirst Platform Foundation for iOS native application for iOS by installing the MobileFirst iOS SDK from the CocoaPods dependency manager, then copying one file from your MobileFirst project to your Xcode project.

Before you begin

- You must have CocoaPods installed in your development environment. For more information, see the "Getting Started" guide for CocoaPods installation.
- You must have already created a MobileFirst native API project with the MobileFirst Platform Command Line Interface for iOS so that you have a client properties (`worklight.plist`) file. For more information, see the “**`create`**” on page 15-6 command.

About this task

The IBM MobileFirst Platform Foundation iOS SDK consists of a collection of pods, available through CocoaPods, that you can add to your project. The pods correspond to core and other functions that are exposed by IBM MobileFirst Platform Foundation for iOS. The SDK contains the following pods for development of new applications:

- `IBMMobileFirstPlatformFoundation`: The core of the system. It implements client/server connections, and handles security, calling adapters, and other required functions.
- If you are using the latest interim fix for IBM MobileFirst Platform Foundation for iOS, `IBMMobileFirstPlatformFoundationJSONStore`: Contains the JSONStore feature. Specify this pod in your podfile if you intend to use the JSONStore feature in your app.
- `CloudantToolkitLocal`: Enables interaction with Cloudant data stores. Specify this pod in your podfile if you intend to use Cloudant in your app for local or remote database storage.

Note:

The following pods are also part of IBM MobileFirst Platform Foundation iOS SDK, but are for use in migrating existing client applications that were developed for IBM MobileFirst Platform for iOS on IBM Bluemix to IBM MobileFirst Platform Foundation for iOS.

- IMFCompatibility: Simplifies the migration process by providing a compatibility API.
- IMFDataLocal: Provides security integration between IBM MobileFirst Platform Foundation for iOS and the Cloudant Toolkit.

For more information about migrating apps from Bluemix Mobile Services see "Migrating applications created on IBM Bluemix to IBM MobileFirst Platform Foundation for iOS" on page 9-1.

The IMFCompatibility, IMFDataLocal, and CloudantToolkitLocal pods all depend on the IBMMobileFirstPlatformFoundation pod, so specifying any one of them causes IBMMobileFirstPlatformFoundation to be included. IMFDataLocal depends on CloudantToolkitLocal, so specifying IMFDataLocal also causes CloudantToolkitLocal to be included. If you are using the latest interim fix for IBM MobileFirst Platform Foundation for iOS, the IBMMobileFirstPlatformFoundationJSONStore pod depends on the IBMMobileFirstPlatformFoundation pod, so specifying IBMMobileFirstPlatformFoundationJSONStore also causes IBMMobileFirstPlatformFoundation to be included.

Procedure

1. Create an Xcode project.
2. Close Xcode.
3. Open **Terminal** at the location of your new Xcode project.
4. Run the **pod init** command to create a Podfile file.
5. Open the Podfile file that is in the root of the project with a text editor.
6. Comment out or remove the existing content.
7. Add the following lines, including the iOS version, and save the changes:

   ```
   source 'https://github.com/CocoaPods/Specs.git'
   use_frameworks!
   target :name-of-the-target-in-xcode-project do
   platform :ios, 9.0
   pod 'IBMMobileFirstPlatformFoundation'
   ```

   Note:

   The pod examples assume you are upgrading to the latest version of the MobileFirst pod. If you are are not using the latest version of MobileFirst, you need to indicate the version. For example, for importing the latest pod for 7.1.0 IBMMobileFirstPlatformFoundation the line would look like this:

   ```
   pod 'IBMMobileFirstPlatformFoundation', '~> 7.1.0'
   ```

8. Optional: Specify additional pods in the file, if your app needs to use the additional functionality that they provide. For example, if your app uses Cloudant for data storage, the podfile might look like:

   ```
   source 'https://github.com/CocoaPods/Specs.git'
   use_frameworks!
   platform :ios, 9.0
   pod 'IBMMobileFirstPlatformFoundation'
   pod 'CloudantToolkitLocal'
   ```

Note: If the podfile in this example specified only the `CloudantToolkitLocal` pod and not the `IBMMobileFirstPlatformFoundation` pod, the results would be the same due to the built-in dependencies.

9. Verify that the Xcode project is closed.
10. Run the **`pod install`** command. This command installs the MobileFirst SDK `IBMMobileFirstPlatformFoundation.framework` and any other frameworks that are specified in the Podfile and their dependencies. It then generates the pods project, and integrates the client project with the MobileFirst SDK.
11. Open your `*ProjectName*.xcworkspace` file in Xcode by typing `open *ProjectName*.xcworkspace` from a command line. This file is in the same directory as the `*ProjectName*.xcodeproj` file.
12. Add a `worklight.plist` client property file to the Xcode project. This file is generated when you create a new IBM MobileFirst Platform Foundation for iOS project with the IBM MobileFirst Platform Command Line Interface for iOS. For more information, see "Client property file" on page 8-10.
    a. Locate the `worklight.plist` in the directory where you ran IBM MobileFirst Platform Command Line Interface for iOS to create the project.
    b. Copy the `worklight.plist` file to a location under the root folder of your Xcode project.

Results

You can now start developing your native iOS application with the IBM MobileFirst Platform Foundation iOS SDK.

Configuring a Swift application

Because Apple Swift is compatible with Objective-C, you can use the MobileFirst API from within an iOS Swift project.

Procedure

1. Create a Swift project and install the MobileFirst SDK into an iOS native application as explained in "Installing required components" on page 9-11.
2. Use the `import IBMMobileFirstPlatformFoundation` statement in any class that needs to use the MobileFirst SDK.

Results

All the MobileFirst classes are now available from any of your Swift files. The Xcode IDE provides code autocompletion, converted to the Swift style.

What to do next

A tutorial is available on the Developer Center. See Configuring a native iOS application with the MobileFirst Platform SDK.

Using Logger in Swift Projects

In order to use `OCLogger` in Swift projects, you can configure your Swift application by following the steps described in this section.

Procedure

1. Create a native MobileFirst application for iOS that uses Swift. For more information, see "Developing native applications" on page 8-7.

2. In the Xcode IDE, create a Swift file and name it `OCLoggerSwiftExtension.swift`.

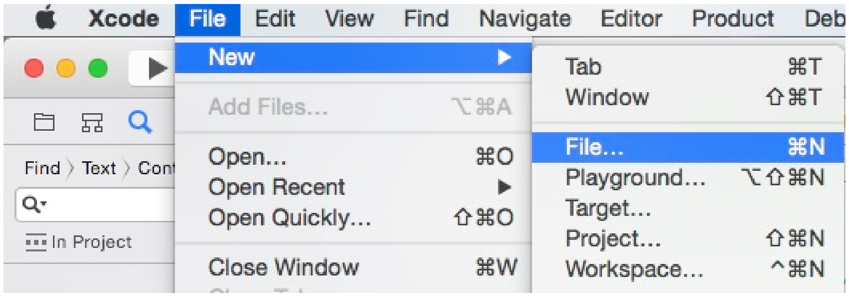


Figure 8-1. New Xcode File

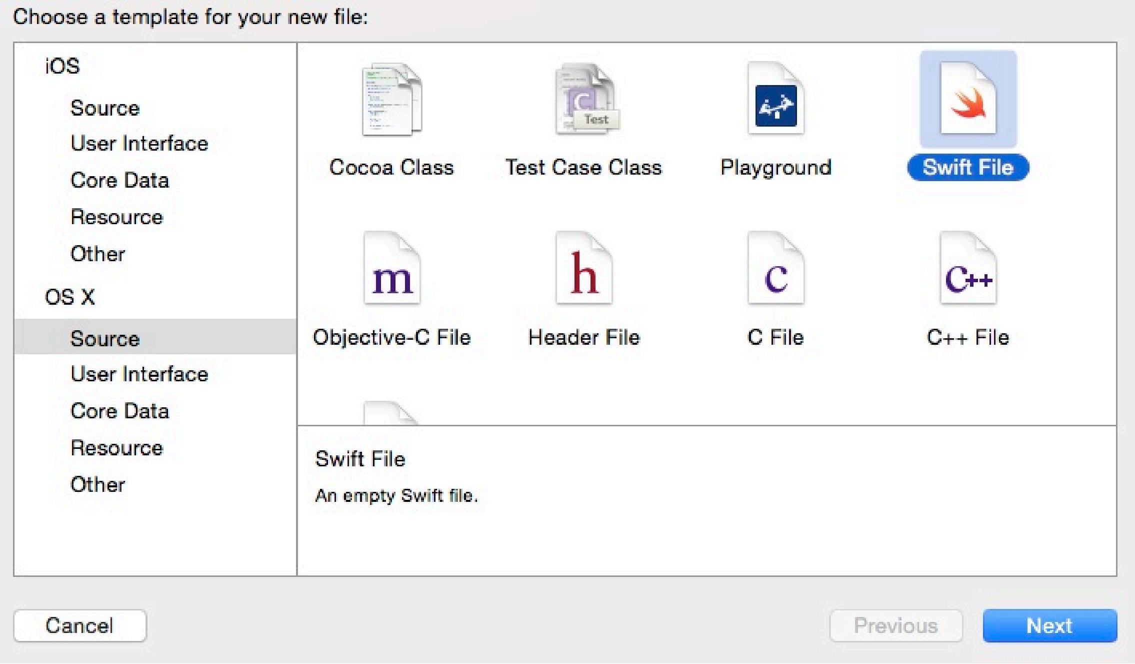


Figure 8-2. Swift File Type

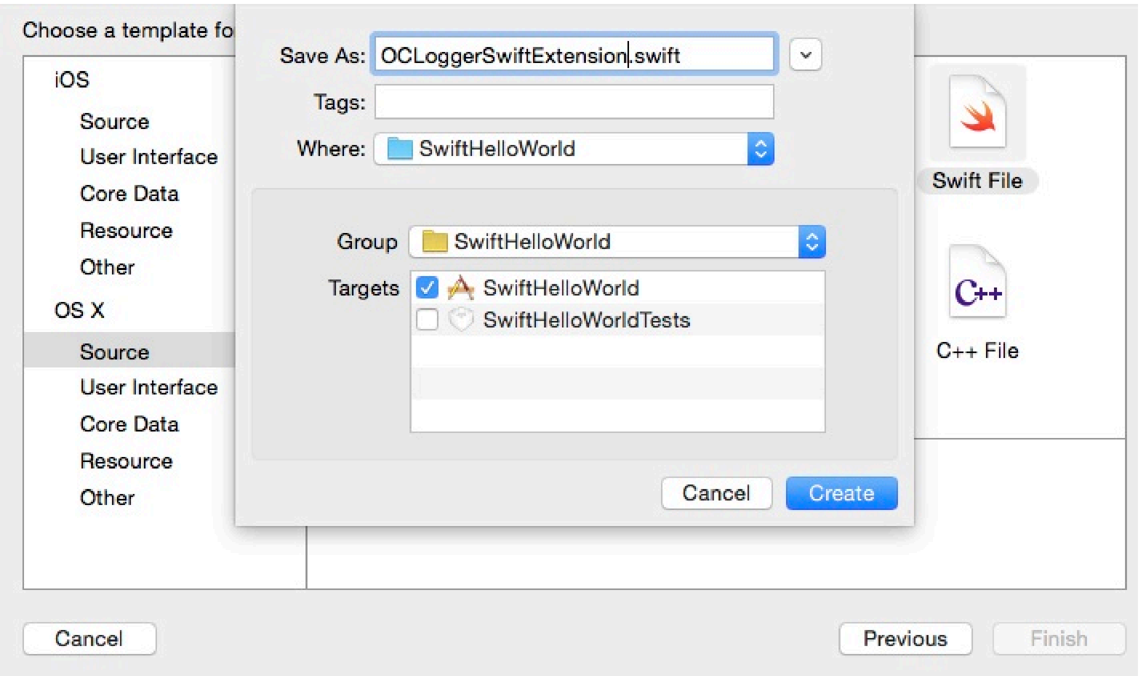


Figure 8-3. Name a Swift File

3. Add the following code to the file:

```
import Foundation

extension OCLogger {
  //Log methods with no metadata

  func logTraceWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logDebugWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logInfoWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logWarnWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logErrorWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logFatalWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  func logAnalyticsWithMessages(message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:Dictionary<String, String>())
  }

  //Log methods with metadata

  func logTraceWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logDebugWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logInfoWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_INFO, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logWarnWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_WARN, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logErrorWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logFatalWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args), userInfo:userInfo)
  }

  func logAnalyticsWithUserInfo(userInfo:Dictionary<String, String>, message:String, _ args: CVarArgType...) {
    logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args), userInfo:userInfo)
  }
}

//Alternatives for macros

private func _metadataDictionary(file:String, fn:String, line:Int) -> Dictionary<String, String> {
return [
    "$method" : fn,
    "$file" : file.lastPathComponent,
    "$line" : String(line),
    "$src": "swift"
    ];
}

public func OCLoggerTrace(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_TRACE, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}

public func OCLoggerDebug(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_DEBUG, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}

public func OCLoggerInfo(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_INFO, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}

public func OCLoggerWarn(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_WARN, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}

public func OCLoggerError(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_ERROR, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}
```

```
public func OCLoggerFatal(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_FATAL, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}

public func OCLoggerAnalytics(message:String, package:String =
"IMF", file: String = __FILE__, fn: String = __FUNCTION__, line: Int = __LINE__, #args: CVarArgType...) {
    OCLogger.getInstanceWithPackage(package).logWithLevel(OCLogger_ANALYTICS, message: message, args:getVaList(args),
userInfo: _metadataDictionary(file, fn, line));
}
```

4. Test that the Swift extension is working by using the Logger in Swift with the following code:

```
OCLogger.setLevel(OCLogger_TRACE)
OCLogger.setCapture(true);

let logger : OCLogger = OCLogger.getInstanceWithPackage("MyTestLoggerPackage")

logger.logTraceWithMessages("Hello %@", "Trace");
logger.logTraceWithMessages("Hello Trace");

OCLoggerDebug("Hello Debug!");
OCLoggerDebug("Hello %@", package: "SomePackageName", args: "Debug!");
```

Results

Verify the output is similar to the following output:

```
SwiftHelloWorld[7591:4265124] [TRACE] [MyTestLoggerPackage] Hello Trace
SwiftHelloWorld[7591:4265124] [TRACE] [MyTestLoggerPackage] Hello Trace
SwiftHelloWorld[7591:4265124] [DEBUG] [IMF] viewDidLoad() in ViewController.swift:26 :: Hello Debug!
SwiftHelloWorld[7591:4265124] [DEBUG] [SomePackageName] viewDidLoad() in ViewController.swift:27 :: Hello Debug!
```

Note: The project name used in this example is `SwiftHelloWorld`. Your project name will show in the output in place of `SwiftHelloWorld`. The code was added to the `viewDidLoad()` function in the `ViewController.swift` file. Your output depends on where you added the code in your project. The timestamps were removed from this example for clarity.

Enforcing TLS-secure connections in iOS apps

From iOS 9 Transport Layer Security (TLS) protocol version 1.2 must be enforced in all apps. You can disable this and bypass the iOS 9 requirement for development purposes.

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.

Starting from the latest interim fix of IBM MobileFirst Platform Foundation for iOS V7.1.0, the apps that you develop in IBM MobileFirst Platform Foundation for iOS

V6.0.0 and later automatically turn off transport security to allow all non-secure connections to the MobileFirst Development Server.

To enable non-TLS connections, 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 IBM MobileFirst Platform Server host name is added automatically to the NSException dictionary. If you are creating a **native** iOS application, however, you must manually add the IBM MobileFirst Platform Server host name to the NSException dictionary.

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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS. To use the MobileFirst 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 MobileFirst Platform Foundation for iOS.

Procedure

On the **Xcode Build Settings** tab, in the **Build Options** group, set **Enable Bitcode** to **No**.

Token licensing: setting the license app type

Set the license application type for your app if your MobileFirst Server uses token licensing.

About this task

If the server on which you are deploying your MobileFirst app has or will have token licensing activated, you must set the license application type so that the correct number of license tokens are used.

You can set the license app type by using the command line, by using MobileFirst Studio, or by manually editing the `application-descriptor.xml` file. The license application type must be one of the following values:

- `NON_PRODUCTION`: Use this value while you are developing and testing the application on the production server.

  Important: Using `NON_PRODUCTION` for a production app is a breach of the license terms.
- `APPLICATION`: Use this value for most applications when you are ready to deploy the app on a production server. This is the default value. The production server must have token licensing activated, and enough tokens must be available.
- `ADDITIONAL_BRAND_DEPLOYMENT`: Use this value when you are ready to deploy an additional brand deployment app on a production server. The app must be an additional brand deployment of an existing app for which an `APPLICATION` token has already been used. Your organization must have purchased an Additional Brand Deployment license for you to use this option. In addition, the code in the similar application must meet specific requirements such as the percentage of code that is different. Refer to your IBM MobileFirst Platform Foundation for iOS - Additional Brand Deployment software license for details.

  The production server must have token licensing activated, and enough tokens must be available.

If token licensing is activated on the server and this element is not present in the application descriptor, then the default setting `APPLICATION` is used.

For more information, see “Licensing in MobileFirst Server” on page 2-10 and your IBM MobileFirst Platform Foundation for iOS software license.

Set the license app type in one of the following ways:

Procedure

- Use the **`mfp config`** command in the MobileFirst Platform Command Line Interface for iOS.

  From a command prompt, run the **`mfp config`** command in either interactive mode or direct mode.
  - Interactive mode:
    1. Type `mfp config`.
    2. Select `Client App Type`.

   3. Select `app_license_type`.
   4. Specify the appropriate license type: `APPLICATION`, `ADDITIONAL_BRAND_DEPLOYMENT`, or `NON_PRODUCTION`.
  – Direct mode:
   1. Type `mfp config app_license_type` *`value`*.

      The *`value`* placeholder takes one of the following values: `APPLICATION`, `ADDITIONAL_BRAND_DEPLOYMENT`, or `NON_PRODUCTION`.
- Use MobileFirst Studio.
  1. Open MobileFirst Studio in your Eclipse workspace.
  2. Double-click the `application-descriptor.xml` file. The application descriptor opens in the Application Descriptor Editor.
  3. In the **Licensing** section of the application descriptor, select the appropriate value for the **License App Type**.
- Manually edit the application descriptor.
  1. Open the `application-descriptor.xml` file for your app in a text editor.
  2. Specify one of the following values for the `licenseAppType` element, as appropriate: `APPLICATION`, `ADDITIONAL_BRAND_DEPLOYMENT`, or `NON_PRODUCTION`.

     Important: If the `licenseAppType` element does not yet exist in the `application-descriptor.xml` file, you must add the `licenseAppType` element as a child of the root element in the file to use token licensing. For more information, see the documentation for the application descriptor for your application type.

Artifacts produced during development cycle

When you use IBM MobileFirst Platform Foundation for iOS 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 iPhone `.ipa` file. This is usually uploaded to an "App Store" such as the Apple Store.

Application metadata and resources (`.wlapp`)
A `.wlapp` file. Metadata and web resources of a MobileFirst application deployed on the MobileFirst Server. Used by the MobileFirst Server to identify and service mobile applications.

Adapter files (`.adapter`)
An adapter file (`.adapter`) contains server-side code written by the MobileFirst developer (for example, retrieve data from a remote database). Adapter code is accessed by MobileFirst 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 11-5.

Developing the server side of a MobileFirst application

This collection of topics relates to various aspects of developing the server-side components of a MobileFirst application.

Overview of MobileFirst adapters

Adapters are the server-side code of applications that are deployed on and serviced by IBM MobileFirst Platform Foundation for iOS.

Adapters run on the IBM MobileFirst Platform Server as middleware between mobile apps and enterprise (back-end) systems.

Adapters are the server-side code of applications that are deployed on and serviced by IBM MobileFirst Platform Foundation for iOS. Adapters connect to enterprise applications (also referred to as back-end systems), deliver data to and from mobile applications, and perform any necessary server-side logic on this data.

You can "hot deploy" adapters, meaning deploy, undeploy, and redeploy them at run time. This capability lends great flexibility to the MobileFirst server-side development process. Here are the benefits of using a Java adapter:

- Ability to fully control the URLs structure, the content types, the request and response headers, content and encoding.
- Easy and fast development and testing by using MobileFirst Studio and CLI.
- Easy and fast deployment to a running MobileFirst Server without compromising on performance and without any downtime.
- Ready-to-use security integration with MobileFirst security that uses simple annotations in the source code.

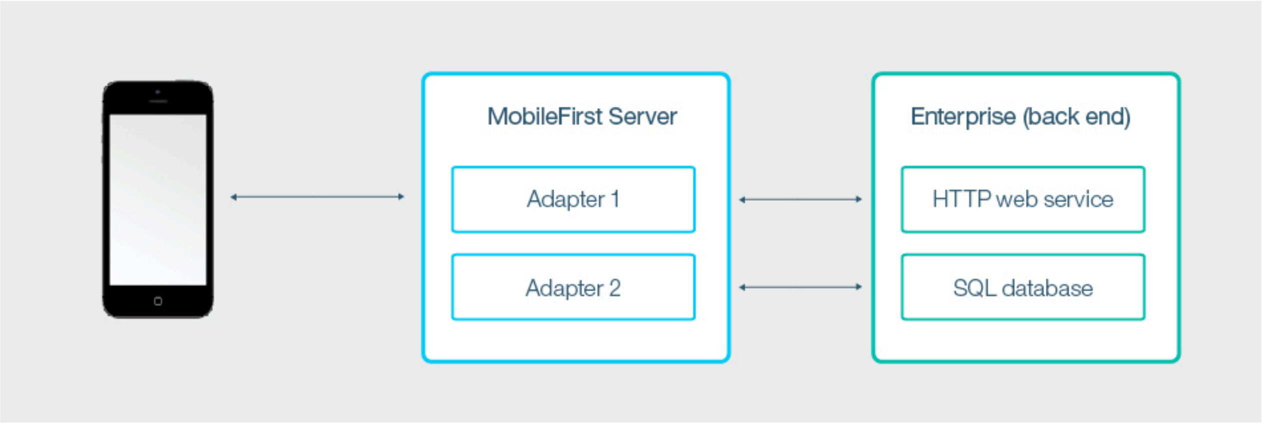


Figure 8-4. MobileFirst adapters

Development Language

Adapters can be developed in either in JavaScript or in Java. The source code of the adapter in both cases has access to a rich server API, which enables it to perform operations on the server side such as sending push notifications, calling other adapters, getting the user identity and more. For more information, see "MobileFirst Java adapters" on page 8-24 and "MobileFirst JavaScript adapters" on page 8-36.

Lifecycle

The adapter lifecycle comprises three stages:

Develop
The adapters are developed in IBM MobileFirst Platform Studio or from the command-line interface (CLI) as a part of the MobileFirst project development.

Test
IBM MobileFirst Platform Studio and CLI tools make it easy to test adapter code. It is also possible to use external tools such as Postman or cURL to test the adapter. The Java adapters tutorials on the Getting Started page of the Developer Center show an implementation of HTTP connectivity in a Java adapter.

Deploy to staging, test, and production environments
After development and testing have been completed, you deploy the adapter to a running MobileFirst runtime of the various runtime environments (beta, test, pre-production, production). You can use the IBM MobileFirst Platform Operations Console to deploy the adapter from its web-based GUI. It is also possible to automate the build and deploy process of MobileFirst adapters by using MobileFirst Apache Ant tasks. For more information, see "Administering MobileFirst applications through Ant" on page 12-12.

Third-party dependencies

MobileFirst Server comes packaged with third-party dependencies. Ensure that you do not package different versions of those dependencies in your project. For a full list of the third-party JAR files in MobileFirst Server, see Third-party adapter dependencies.

MobileFirst Java adapters

With IBM MobileFirst Platform Foundation for iOS, you can create, test, and deploy adapters that are written in Java.

Java adapters were introduced to IBM MobileFirst Platform Foundation for iOS, starting with V7.0. Java adapters, which are written in Java, are based on the JAX-RS specification and make a full REST API available to the client. In other words, a Java adapter is a JAX-RS service that can be deployed to MobileFirst Server and that by default, has access to MobileFirst Server APIs.

Here are the benefits of using a Java adapter:

- Ability to fully control the URLs structure, the content types, the request and response headers, content and encoding.
- Rapid development and testing by using MobileFirst Studio and CLI.
- Rapid deployment to a running MobileFirst Server without compromising on performance and without any downtime.
- Ready-to-use security integration with MobileFirst security that uses simple annotations in the source code.

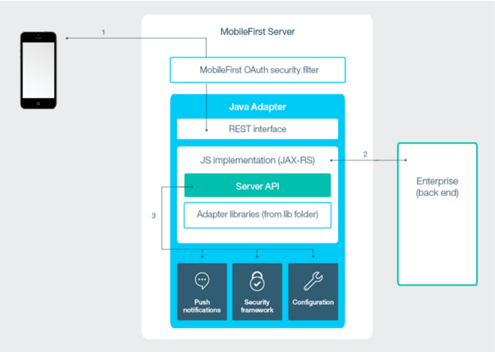


Figure 8-5. The Java adapter framework

Figure 1 illustrates how a mobile device (1) can access any Java adapter from its REST endpoint. The REST interface is protected by the MobileFirst OAuth security filter, meaning that the client needs to obtain an access token to access the adapter resources. Because any of the resources of the adapter can have its own URL, it is possible to use a third-party (that is, non-MobileFirst) URL-based security framework. The REST interface invokes the Java code (JAX-RS service) to handle incoming requests. The Java code can perform operations on the server by using the Java MobileFirst Server API (3). In addition, the Java code can connect to the enterprise system to fetch data, update data, or perform any other operation that the enterprise system exposes (2).

For more information about MobileFirst adapters, see “Overview of MobileFirst adapters” on page 8-23.

Adapter sandboxing

Every Java adapter has its own isolated sandbox, in which all its classes are running without knowing about or interrupting the sandboxes of other adapters. It is possible to include third-party libraries that are required by the adapter code in the `adapter/lib` folder. Libraries that are included in that folder override any libraries that are provided by the application server or included in the `server/lib` folder.

Setting up the connectivity to back-end configuration

Unlike JavaScript adapters, Java adapters do not provide back-end connectivity, by default. It is the responsibility of the adapter developer to write code that connects to the back-end system. Examples of open source libraries are Apache HttpClient for connecting over HTTP and java.sql APIs for connecting directly to databases.

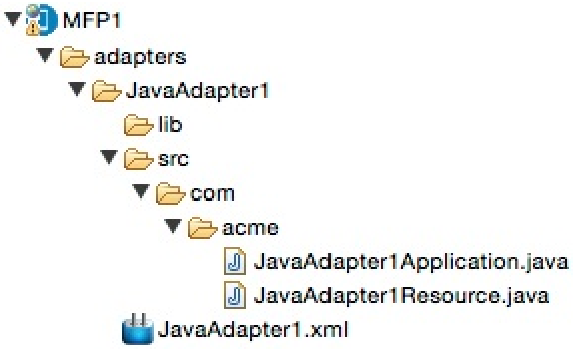
For more information about connectivity and samples of implementations of back-end connectivity inside a Java adapter, see Java Adapter.

Structure of Java adapters

Learn about the make-up of the adapter in IBM MobileFirst Platform Studio and about the adapter configuration in the XML adapter file.

In Studio

Java adapter structure is shown in IBM MobileFirst Platform Studio, as follows:



There are two folders under each Java adapter:

- **src**: This folder must contain the Java sources of the JAX-RS service. JAX-RS services are made up of an application class, which extends javax.ws.rs.core.Application and resource classes.
- **lib**: This folder can be used for JAR library files that are available to the adapter during runtime, for example, if you are using a third-party library that is specific to the current adapter.

  Note: These JARs are loaded according to a "parent-last" policy, meaning that they override any libraries that were provided by the application server.

The Adapter XML configuration file

The Java adapter has an XML configuration file, as in all types of MobileFirst adapters.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<wl:adapter name="JavaAdapter1"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:wl="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">
    <displayName>JavaAdapter1</displayName>
    <description>JavaAdapter1</description>
    <connectivity>
        <connectionPolicy xsi:type="wl:NullConnectionPolicyType"></connectionPolicy>
    </connectivity>

    <JAXRSApplicationClass>com.acme.JavaAdapter1Application</JAXRSApplicationClass>
</wl:adapter>
```

Note the following points:

- The new <JAXRSApplicationClass> element defines the class name of the JAX-RS application of this adapter. In the previous example, it is com.acme.JavaAdapter1Application. For more information, see “Implementing the JAX-RS service of the adapter.”
- Java adapters have been designed to give maximum flexibility to the developer, but connection policies that are provided in JavaScript adapters are not available for Java adapters. For this reason, the NullConnectionPolicy definition for the <connectivity> element must remain as-is. It is the responsibility of the developer to implement the connection to the back end, as required. The Java Adapter tutorial in Getting Started shows an implementation of HTTP connectivity in a Java adapter.

Developing Java adapters

Learn about creating a new Java adapter and developing Java adapter code.

Creating a Java adapter using MobileFirst CLI commands:

Follow these instructions to create a Java adapter usingMobileFirst CLI commands.

About this task

You can create a Java adapter from the command line.

Procedure

1. Open a terminal.
2. Change directory to the MobileFirst project directory.
3. Type: **`mfp add adapter <name> -type java -package <package name>`**.

Developing Java adapter code:

Learn about implementing the adapter's JAX-RS service, the Java server side API, and debugging Java adapter code.

Implementing the JAX-RS service of the adapter:

To implement the JAX-RS service of the adapter, you must first implement the JAX-RS application class, then implement the JAX-RS resources classes.

Overview:

- “Implementing the JAX-RS application class”
- “Implementing a JAX-RS resource” on page 8-28
- “Security configuration of a JAX-RS resource” on page 8-30

Implementing the JAX-RS application class:

About this task

The JAX-RS application class tells the JAX-RS framework which resources are included in the application. Any resource can have a separate set of URLs. Traditionally the application class should extend javax.ws.rs.core.Application and implement the method getClasses or getSingletons that will be called by the JAX-RS framework to get information about this application. In the following example, a JAX-RS application defines three resources: Resource1, UsersResource, and MyResourceSingleton. The first two are provided by the getClasses method, while the last is provided by getSingletons:

```
import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;

public class MyApplication extends Application{

    @Override
    public Set<Class<?>> getClasses() {
        HashSet<Class<?>> classes = new HashSet<Class<?>>();
        classes.add(Resource1.class);
        classes.add(UsersResource.class);
        return classes;
    }

    @Override
    public Set<Object> getSingletons() {
        Set<Object> singletons = new HashSet<Object>();
        singletons.add(MyResourceSingleton.getInstance());
        return singletons;
    }
}
```

An alternative to using the javax.ws.rs.core.Application is to use com.worklight.wink.extensions.MFPJAXRSApplication. In the following example, it is not necessary to put all the resources classes (or singletons) in a list, because MFPJAXRSApplication scans the package that is returned by the getPackageToScan method for JAX-RS resources and creates the list automatically. Additionally, its init method is called by IBM MobileFirst Platform Server as soon as the adapter is deployed (before serving has begun) and when the MobileFirst runtime starts up:

```
import com.worklight.wink.extensions.MFPJAXRSApplication;
public class MyApplication2 extends MFPJAXRSApplication{
    @Override
    protected void init() throws Exception {
        //Perform initialization
    }

    @Override
    protected String getPackageToScan() {
        return getClass().getPackage().getName();
    }
}
```

Implementing a JAX-RS resource:
About this task

A JAX-RS resource is a POJO (plain old Java object) which is mapped to a root URL and has Java methods for serving requests to this root URL and its sub-URLs. For example:

```
import java.util.ArrayList;
import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

@Path("/users")
//This is the root URL of the resource ("/users")
public class UsersResource {
    //Instead of this static field, it could be a users DAO that works with Database or cloud storage
    static ArrayList<User> users = new ArrayList<User>();

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    //This will serve: GET /users
    public ArrayList<User> getUsers(){
        return users;
    }

    @Path("/{userId}")
    @Produces(MediaType.APPLICATION_JSON)
    //This will serve: GET /users/{userId}
    public User getUser(@PathParam("userId") String userId){
        return findUserById(userId);
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    //This will serve: POST /users
    public void addUser(User u) {
        users.add(u);
    }

    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    //This will serve: PUT /users
    public Response updateUser(User u) {
        User user = findUserById(u.getId());
        if (user == null){
            return Response.status(Status.NOT_FOUND)
                           .entity("User with ID: "+u.getId()+" not found")
                           .build();
        }
        users.remove(user);
        users.add(u);
        return Response.ok().build();
    }

    @DELETE
    @Path("/{userId}")
    //This will serve: DELETE /users/{userId}
    public void deleteUser(@PathParam("userId") String userId){
        User user = findUserById(userId);
        users.remove(user);
    }

    private User findUserById(String userId) {
        //TODO implement...
        return null;
    }
}
```

The resource just shown is mapped to the URL /users and serves the following requests:

Table 8-2. Resource requests.

Request	Description
GET /users	Gets all users list
POST /user	Adds a new user
GET /users/{id}	Gets a specific user with id **id**
PUT /users	Updates an existing user
DELETE /users/{id}	Deletes a user with id **id**

The JAX-RS framework does the mapping from the simple Java object User to a JSON object and conversely, making it easier for the service developer to use without taking care of repeating conversion-related code. The implementation also helps in extracting parameter values from the URL and from the query string without having to parse it manually.

Security configuration of a JAX-RS resource:

About this task

A JAX-RS adapter resource is protected by default by the MobileFirst security framework, meaning that access to the resource requires a valid OAuth access token. You can configure the resource protection by using the @OAuthSecurity annotation of the MobileFirst Java server-side com.worklight.core.auth package to assign a custom security scope, or to disable resource protection. The annotation can be applied either to a specific resource method or to an entire resource class. For detailed usage instructions, see the annotation's API reference.

Examples

Using @OAuthSecurity at the resource class level

To disable the default protection of the UsersResource class that was used in the previous examples, and make all resource procedures in this class unprotected, add the OAuthSecurity annotation to the class definition, and set the annotation's enabled element to `false`:

```
@OAuthSecurity(enabled=false)
```

The annotation should be added above the class definition, as shown in the following example:

```
@Path("/users")
@OAuthSecurity(enabled=false)
//This is the root URL of the resource ("/users")
public class UsersResource {
...
```

Using @OAuthSecurity at the method level

Suppose that you want to disable protection of the user's resource for read operations, but protect the add, edit, and delete user operations. To do so, add the @OAuthSecurity annotation with the enabled element set to false at the class level, and then add the annotation to the methods that you want to protect to override the class annotation. In the following example, the addUser, updateUser, and deleteUser methods are protected by an `adminRealm` scope:

```
@Path("/users")
@OAuthSecurity(enabled=false)
//This is the root URL of the resource ("/users")
public class UsersResource {

    @GET
    @Produces(MediaType.APPLICATION_JSON)
    //This will serve: GET /users
    public ArrayList<User> getUsers(){
        ...
    }

    @Path("/{userId}")
    @Produces(MediaType.APPLICATION_JSON)
    //This will serve: GET /users/{userId}
    public User getUser(@PathParam("userId") String userId){
       ...
    }

    @POST
    @Consumes(MediaType.APPLICATION_JSON)
    @OAuthSecurity(scope="adminRealm")
    //This will serve: POST /users
    public void addUser(User u) {
       ...
    }

    @PUT
    @Consumes(MediaType.APPLICATION_JSON)
    @OAuthSecurity(scope="adminRealm")
    //This will serve: PUT /users
    public Response updateUser(User u) {
        ...
    }

    @DELETE
    @Path("/{userId}")
    @OAuthSecurity(scope="adminRealm")
    //This will serve: DELETE /users/{userId}
    public void deleteUser(@PathParam("userId") String userId){
        ...
    }

    ...
}
```

Java server side API:

Java adapters can use the IBM MobileFirst Platform Server Java API to perform server-related operations such as: calling other adapters, submitting push notifications, logging on to the server log, getting values of configuration properties, reporting activities to analytics, and getting the identity of the request issuer.

To get the server API interface use the following code:

```
WLServerAPI serverApi = WLServerAPIProvider.getWLServerAPI();
```

The WLServerAPI interface is the parent of all the API categories: analytics, configuration, push, adapters, authentication. If you want to use the push API, you can write:

```
PushAPI pushApi = serverApi.getPushAPI();
```

For more information on the Java API, see the . Examples of the use of the API are provided in the following sections.

Logger

Adding a logger to the JAX-RS resource of your adapter is essential for logging debug information, information, warnings, and errors to the MobileFirst Server log. In the Java adapters, you can use the standard `java.util.logging` logger. To add logger declare the following static field:

```
static Logger logger = Logger.getLogger(UsersResource.class.getName());
```

Note: The logger is part of the new adapter template, so in most cases, there is no need to add it.

Push API

You can extend the UsersResource described in “Implementing the JAX-RS service of the adapter” on page 8-27 to notify all the devices of a user when the user record is updated:

```
WLServerAPI serverAPI = WLServerAPIProvider.getWLServerAPI();

@PUT
@Consumes(MediaType.APPLICATION_JSON)
public Response updateUser(User u) {

    //TODO Really update the user...

    //Notify all devices that the user has been updated
    PushAPI push = serverAPI.getPushAPI();
    INotification notification = push.buildNotification();
    notification.getTarget().setUserIds(u.getId());//Targeted only for the updated user

    JSONObject payload = new JSONObject();
    payload.put("user", u.getId());
    notification.getSettings().getAPNS().setPayload(payload);

    try {
        push.sendMessage(notification, "myApp");
    } catch (MFPSendMessageFailedException e) {
        e.printStackTrace(); //Ignore failure in this case, since the update succeeded.
    }
    return Response.ok().build();
}
```

Security API

The security API provides the security context object that contains information about the identity of the request issuer. In the following example, a new validation is added to the user's resource: User records can be updated only by users themselves:

```
WLServerAPI serverAPI = WLServerAPIProvider.getWLServerAPI();

@PUT
@Consumes(MediaType.APPLICATION_JSON)
publis Response updateUser(User u) {

    SecurityAPI security = serverAPI.getSecurityAPI();
    String requestIssuerUserId = security.getSecurityContext().getUserIdentity().getId();
    if (!u.getId().equals(requestIssuerUserId)){
        return Response.status(Status.FORBIDDEN)
                      .entity("The user: "+requestIssuerUserId+
                              " is not allowed to modify the user record of another user").build();
    }

    //TODO Really update the user...
```

Adapters API

The adapters API makes it easy to perform requests to other adapters in the same server.

The following example shows how to use the adapters API to call a JavaScript adapter:

```
@Path("/")
public class JavaAdapter1Resource {
    static Logger logger = Logger.getLogger(JavaAdapter1Resource.class.getName());

    @GET
    @Path("/calljs")
    @Produces("application/json")
    public JSONObject callJSAdapterExample() throws Exception{
        AdaptersAPI adaptersAPI = WLServerAPIProvider.getWLServerAPI().getAdaptersAPI();
        //Using helper method to create a request to the JS adapter
        HttpUriRequest req = adaptersAPI.createJavascriptAdapterRequest("JSAdapter", "getStories");
        //Execute the request and get the response
        HttpResponse resp = adaptersAPI.executeAdapterRequest(req);
        //Convert the response to JSON since we know that JS adapters always return JSON
        JSONObject json = adaptersAPI.getResponseAsJSON(resp);
        //Return the json response as the response of the current request that is being taking care of
        return json;
    }
```

Configuration API

The configuration API enables the adapter to read server-side configuration properties defined in the `worklight.properties` file or as JNDI entries. For more information about MobileFirst configuration properties, see “CLI Commands” on page 15-1.

For example, to get the value of the serverSessionTimeout property, write the following code:

```
String serverSessionTimeout = serverAPI.getConfigurationAPI().getMFPConfigurationProperty("serverSessionTimeout");
```

Saving applicative state between requests in Java RESTful adapters

In versions earlier than V7.1.0, developers were able to store the applicative state in the HTTP session, by using the session object, namely request.getSession() (see WL.Server).

If you are working in session-independent mode that became available starting with IBM MobileFirst Platform Foundation for iOS V7.1.0, the applicative state of the adapter must be persisted outside the session, for example, by using a database such as Cloudant. For more information on session-independent mode, including steps for upgrading projects that were created in earlier versions of IBM MobileFirst Platform Foundation for iOS, see “Session-independent mode” on page 8-108.

The following code snippets provide examples for migrating code from an earlier version of IBM MobileFirst Platform Foundation for iOS so that it support the changes for session-independent mode in V7.1.0.

Consider a shopping-cart app, in which users can add items to a cart, see the list of items that are already in the cart, and submit an order when satisfied with the items in it.Before V7.1.0, this functionality might have been implemented in the following code:

```
//Define the server API to be able to perform server operations
WLServerAPI api = WLServerAPIProvider.getWLServerAPI();

@GET
@Path("/addItemToCart")
public void addItemToCart(@Context HttpServletRequest request, @HeaderParam("item") String item) {
    String oldItems = getItemsFromCart(request);

    HttpSession session = request.getSession();
    session.setAttribute("items", oldItems + " " + item);

}

@GET
@Path("/getItemsFromCart")
public String getItemsFromCart(@Context HttpServletRequest request) {
    HttpSession session = request.getSession();
    return (String) session.getAttribute("items");
}
```

Assume you are using Cloudant as the persist store over the Cloudant Java Client API and that you are implementing a Java RESTful HTTP adapter called ShoppingCartAdapter. See Figure 1: using OAuthSecurityContext.getClientId(), the adapter requests the ID that is associated with the current client from MobileFirst Serverand uses it when reading or writing applicative data to or from Cloudant.

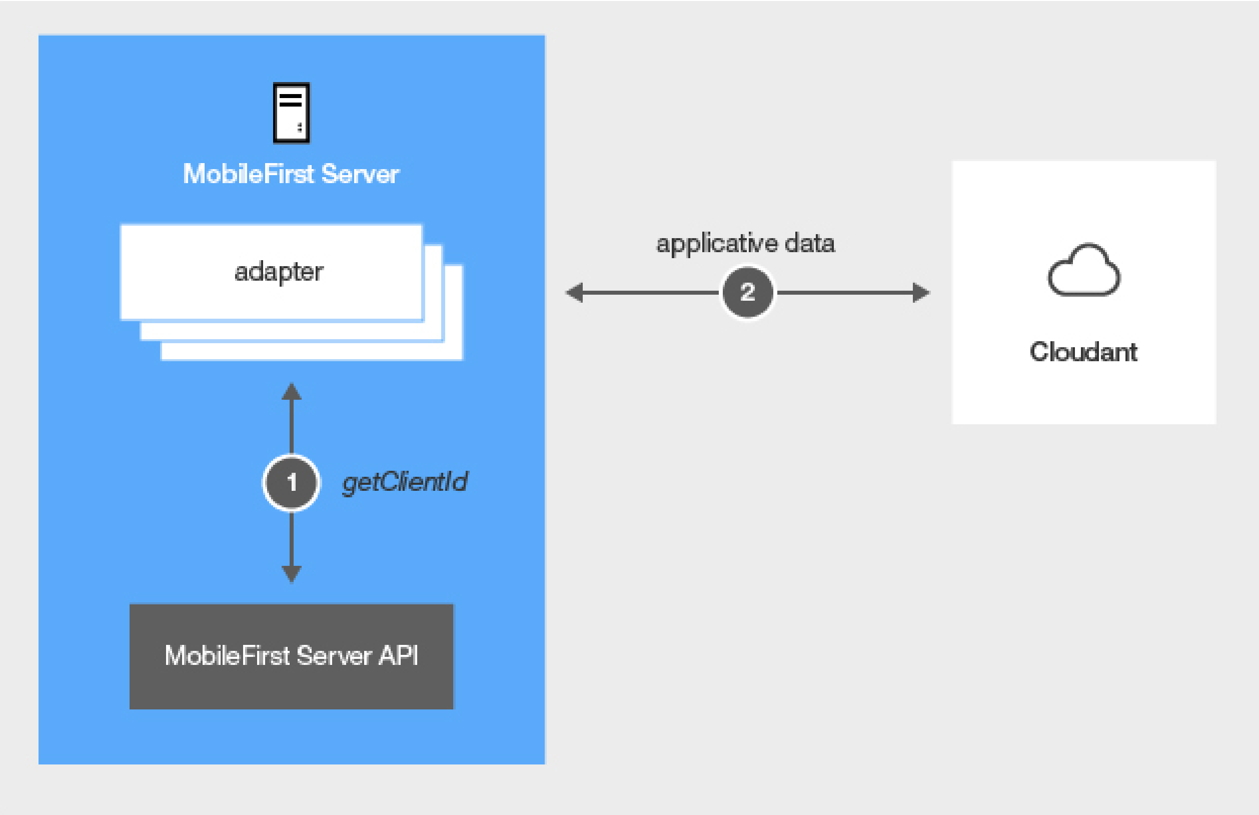


Figure 8-6. Retrieving applicative data using the server-side OAuthSecurityContext.getClientId API.

If you adapt the code to work in session-independent mode under V7.1.0, the code for MobileFirst V7.1.0 might look as follows:

```
//Define the server API to be able to perform server operations
WLServerAPI api = WLServerAPIProvider.getWLServerAPI();

CloudantClient client = new CloudantClient("cloudantId", "username", "password");
@GET
@Path("/addItemToCart")
```

```
public void addItemToCart(@HeaderParam("item") String item) {
    // Get the clientID from the MFP API (new in 7.1.0)
    String userId = api.getSecurityAPI().getSecurityContext().getClientId();
    saveItemToCloudant("shoppingcartdb", userId, item);
    }

@GET
@Path("/getItemsFromCart")
public String getItemsFromCart() {
    // Get the clientID from the MFP API (new in 7.1.0)
    String userId = api.getSecurityAPI().getSecurityContext().getClientId();
    return (String) readItemFromCloudant("shoppingcartdb", userId).get("items");
}

/**
 * Reads a JSONObject shopping cart item from Cloudant and returns the document
 * that corresponds to this _id
 * @param db
 * @param key
 * @return a JSONObject
 */
public JSONObject readItemFromCloudant(String db, String key){
    Database cloudantDb = client.database(db, false);

    // Search for the document
    return cloudantDb.find(JSONObject.class, key);
}

/**
 * Saves a JSONObject shopping cart item to Cloudant
 * @param db - the DB to save to
 * @param key - the _id
 * @param value
 */
public void saveItemToCloudant(String db, String key, String value){
    Database cloudantDb = client.database(db, false);

    // Search for the document
    try{
        JSONObject document = cloudantDb.find(JSONObject.class, key);
        String listOfItems = (String) document.get("items");
        // Add new shopping cart item to list
        listOfItems += " " + value;
        document.put("items", listOfItems);

        // Update the document
        cloudantDb.update(document);
    }catch(NoDocumentException e){
        // New Document
        JSONObject obj = new JSONObject();
        obj.put("_id", key);
        obj.put("items", value);

        // Save the object to cloudant db
        cloudantDb.save(obj);
    }
```

Debugging Java adapter code:

Use IBM MobileFirst Platform Studio's debug mode to debug Java adapter code.

Starting debug mode:

Before you begin

After you enter debug mode, you can debug the Java code exactly as you debug a standard Java application. You might need to issue a request to the adapter to make its code run and hit the breakpoints.

Note: While in debug mode, you can debug the code of all deployed Java adapters and the code in the **server/java** folder as well.

Procedure

1. Right-click a Java adapter and select **Debug As** > **Debug MobileFirst Java Adapters**.
2. Optional: Open the **Debug** perspective in MobileFirst Studio to see threads, breakpoints, and values.

Stopping debug mode:

Procedure

1. Open the **Debug** perspective in MobileFirst Studio.
2. Right-click the **Debug** tab and select **remote debug [Remote Java Application]** > **Terminate**.

MobileFirst JavaScript adapters

With IBM MobileFirst Platform Foundation for iOS, you can create, test, and deploy adapters that are written in JavaScript.

You can create and configure all JavaScript adapters manually. In addition, you can automatically generate SAP Netweaver Gateway or SOAP adapters with the services discovery wizard.

Starting with IBM MobileFirst Platform Foundation for iOS V6.3, changes were made to the XML definition and behavior of JavaScript adapter timeout and concurrency. For more information, see “Adapter timeout and concurrency” on page 8-66.

For more general information about IBM MobileFirst Platform Foundation for iOS adapters, see “Overview of MobileFirst adapters” on page 8-23.

Benefits of MobileFirst JavaScript adapters

JavaScript 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**: MobileFirst adapters support read-only and transactional access modes to back-end systems.
- **Security**: MobileFirst 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 that is retrieved from back-end applications is available in a uniform manner, so that application developers can access data uniformly, regardless of its source, format, and protocol.

- **REST Interface**: With the new REST interface, you can benefit from the OAuth 2.0 security framework, even when you use your existing JavaScript adapters. For more information, see "Accessing adapters from the `/adapters` endpoint" on page 8-118.

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 MobileFirst Server are supplied by the adapter or app developer. The procedure and auto-conversions are part of IBM MobileFirst Platform Foundation for iOS.

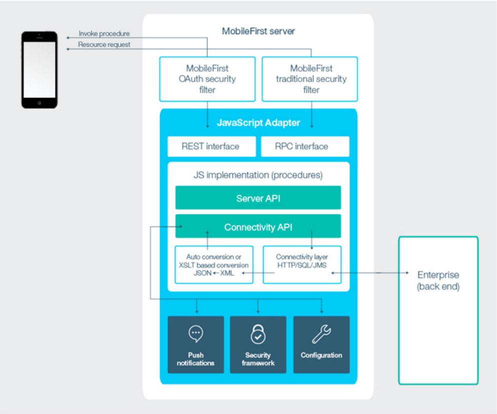


Figure 8-7. The JavaScript adapter framework

1. An adapter provides 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 MobileFirst Server keeps the data intact.

- If this format is not JSON, the MobileFirst 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 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.
   - Take note of the following points:
   - 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.
   - HTTP POST requests are used for client-server communications between the MobileFirst application and the MobileFirst 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 JavaScript adapters

MobileFirst 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 provided 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 that is 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. For more information, see “Structure of the adapter XML file.”
- `<procedure>`: Declares a procedure that is provided by the adapter. For more information, see “Implementing adapter procedures” on page 8-94.

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>
```

Structure of the adapter XML file

The adapter XML file is used to configure connectivity to the back-end system and to declare the procedures that are exposed by the adapters to applications and to other adapters.

The adapter element

The adapter element is the root element of the adapter configuration file. It has the following structure, which consists of both attributes and subelements:

```
<wl:adapter name="Java_adap"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.ibm.com/mfp/integration"
  xmlns:http="http://www.ibm.com/mfp/integration/http">

  <displayName>Java_adap</displayName>
  <description>Java_adap</description>
  <connectivity>
    <connectionPolicy xsi:type="wl:NullConnectionPolicyType"></connectionPolicy>
  </connectivity>

  <JAXRSApplicationClass>java_package.Java_adapApplication</JAXRSApplicationClass>
</wl:adapter>
```

The adapter element attributes

IBM MobileFirst Platform Foundation for iOS provides two schemas that are used by all adapters. Additionally, this product provides a specific schema for each type of adapter. Each schema must be associated with a different namespace. Namespaces are declared by using the xmlns attribute, and are linked to their schemas by using the xsi:schemaLocation attribute. The <adapter> element contains the following attributes.

name

Mandatory.

The name of the adapter. This name must be unique within the MobileFirst Server. It can contain alphanumeric characters and underscores, and must start with a letter. After you define and deploy an adapter, you cannot modify its name.

xmlns:*namespace*

Mandatory.

Defines schema *namespaces*.

This attribute must appear three times.

- xmlns:xsi: Defines the namespace associated with the http://www.w3.org/2001/XMLSchema-instance schema.
- xmlns:wl: Defines the namespace that is associated with the http://www.ibm.com/mfp/integration schema.
- xmlns:*namespace*: Defines the namespace that is associated with the schema related to the back-end application. For example, xmlns:sap or xmlns:sql.

xsi:schemaLocation

Optional.

Identifies the schema locations.

```
xsi:schemaLocation="http://www.ibm.com/mfp/integration location-of-integration-schema-file
URI-of-specific-adapter-schema location-of-schema"
```

If this attribute is missing, auto-complete for XML elements and attributes that are defined win the schema is not available in MobileFirst Studio.

At run time, this attribute has no effect.

The `adapter` element subelements

The `adapter` element has the following subelements.

`displayName`

Optional.

This element is deprecated. The name of the adapter that is displayed in the MobileFirst Operations Console. If this element is not specified, the value of the `name` attribute is used instead.

`description`

Optional.

Additional information about the adapter. Displayed in the MobileFirst Operations Console.

`connectivity`

Mandatory.

Defines the mechanism by which the adapter connects to the back-end application. It contains the `connectionPolicy` subelement. The `connectionPolicy` subelement is mandatory, and it defines connection properties. The structure of this subelement depends on the integration technology of the back-end application.

`procedure`

Mandatory.

Defines a process for accessing a service that is exposed by a back-end application. Occurs once for each procedure that is exposed by the adapter.

The `procedure` element has the following structure.

```
<procedure
  name="unique-name"
  connectAs="value"
  audit="value"
  securityTest="value"
/>
```

The `procedure` element has the following attributes.

`name`

Mandatory.

The name of the procedure. This name must be unique within the adapter. It can contain alphanumeric characters and underscores, and must start with a letter.

`connectAs`

Optional.

Defines how to create a connection to the back end for invoking the retrieve procedure. The following values are valid.

- `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 you identify a user realm in the security tests for this procedure.

audit

Optional.

Defines whether calls to the procedure are logged in the audit log. The log file is *Project Name*/server/log/audit/audit.log.

The following values are valid.

- 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.

Cast Iron adapter connectionPolicy element:

You can access various enterprise data sources, such as databases, web services, and JMS, and provides validation, aggregation, and formatting capabilities by using Cast Iron.

Structure

The connectionPolicy element has the following structure.

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
  <protocol>http</protocol>
  <domain>your.CastIron.com</domain>
  <port>80</port>
  <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
  <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
  <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
</connectionPolicy>
```

Attributes

The connectionPolicy element has the following attributes.

xsi:type

Mandatory.

The value of this attribute must be set to http:HTTPConnectionPolicyType.

Subelements

The connectionPolicy element has the following subelements.

protocol

Optional.

The URL protocol to use. The following values are valid.

- http: default
- https

domain

Mandatory.

The host address.

port

Optional.

The port address.

connectionTimeoutInMilliseconds

Optional

The time until a connection to the back-end can be established. This timeout does not ensure that a timeout exception occurs after a time elapses after the invocation of the HTTP request.

If you pass a different value for this parameter in the `invokeHTTP()` JavaScript function, you can override the value that is defined here. For more information, see the WL.Server class.

socketTimeoutInMilliseconds

Optional.

The timeout between two consecutive packets, starting from the connection packet. This timeout does not ensure that a timeout exception occurs after a time elapses after the invocation of the HTTP request.

If you pass a different value for this parameter in the `invokeHttp()` JavaScript function, you can override the value that is defined here. For more information, see the WL.Server class.

maxConcurrentConnectionsPerNode

Optional.

Defines the maximum number of concurrent connections, which the MobileFirst Server can open to the back end.

IBM MobileFirst Platform Foundation for iOS does not limit the incoming service requests from applications. This subelement can be configured at the application server level. This product limits only the number of concurrent HTTP connections to the back-end service.

The default number of concurrent HTTP connections is 50. You can modify this number based on the expected concurrent requests to the adapter and the maximum requests allowed on the back-end service. You can also configure the back-end service to limit the number of concurrent incoming requests.

Consider a two-node system, where the expected load on the system is 100 concurrent requests and the back-end service can support up to 80 concurrent requests. You can set `maxConcurrentConnectionsPerNode` to 40. This setting ensures that no more than 80 concurrent requests are made to the back-end service.

If you increase the value, the back-end application needs more memory. To avoid memory issues, do not to set this value too high. Instead, estimate the average and peak number of transactions per second, and evaluate their average duration. Then, calculate the number of required concurrent connections as indicated in this example, and add a 5-10% margin. Then, monitor your back end, and adjust this value as required, to ensure that your back-end application can process all incoming requests.

When you deploy 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 the **Scalability and Hardware Sizing** document and the hardware calculator spreadsheet at Developer Center website for IBM MobileFirst Platform Foundation.

HTTP adapter `connectionPolicy` element:

You can use the HTTP adapter to start RESTful services and SOAP-based services. You can also complete HTML scraping.

Structure

The `connectionPolicy` element has the following structure.

```
<connectionPolicy xsi:type="http:HTTPConnectionPolicyType"; cookiePolicy="cookie-policy" maxRedirects="int">
  <protocol>protocol</protocol>
  <domain>host-name</domain>
  <port>host-port</port>
  <connectionTimeoutInMilliseconds>connection_timeout</connectionTimeoutInMilliseconds>
  <socketTimeoutInMilliseconds>socket_timeout</socketTimeoutInMilliseconds>
  <authentication> ... </authentication>
  <proxy> ... </proxy>
  <sslCertificateAlias>ssl-certificate-alias</sslCertificateAlias>
  <sslCertificatePassword>ssl-certificate-password</sslCertificatePassword>
  <maxConcurrentConnectionsPerNode>max_concurrent_connections</maxConcurrentConnectionsPerNode>
</connectionPolicy>
```

Attributes

The `connectionPolicy` element has the following attributes.

`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. The following values are valid.

- `RFC_2109`: default value
- `RFC_2965`
- `NETSCAPE`
- `IGNORE_COOKIES`

For more information about these values, see HTTP components.

`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. If this 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. The default value is `10`.

Subelements

The `connectionPolicy` element has the following subelements.

`protocol`

Optional.

The URL protocol to use. The following values are valid.

- `http`: default
- `https`

`domain`

Mandatory.

The host address.

`port`

Optional.

The port address.

`sslCertificateAlias`

Optional for regular HTTP authentication and simple SSL authentication.

Mandatory for mutual SSL authentication.

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. For more information about the keystore setup process, see "SSL certificate keystore setup" on page 11-59.

`sslCertificatePassword`

Optional for regular HTTP authentication and simple SSL authentication.

Mandatory for mutual SSL authentication.

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. For more information about the keystore setup process, see "SSL certificate keystore setup" on page 11-59.

`authentication`

Optional.

Authentication configuration 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 the `<authentication>` element in one of the following ways.

- Basic Authentication

```
<authentication>
  <basic/>
</authentication>
```

- Digest Authentication

```
<authentication>
  <digest/>
</authentication>
```

- NTLM Authentication

```
<authentication>
  <ntlm workstation="value"/>
</authentication>
```

The `workstation` attribute is optional. This attribute denotes the name of the computer on which the MobileFirst Server runs.

- SPNEGO/Kerberos Authentication

```
<authentication>
  <spnego stripPortOffServiceName="true"/>
</authentication>
```

The attribute `stripPortOffServiceName` is optional. This attribute specifies whether the Kerberos client uses the service name without the port number.

When you use this option, you must also place the `krb5.conf` file under *`Project Name`*`/server/conf`. This 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.

If the adapter exposes procedures with the attribute `connectAs="server"`, the connection policy can contain `<serverIdentity>` element. This feature applies to all authentication schemes. For example:

```
<authentication>
  <basic/>
  <serverIdentity>
    <username> ${DOMAIN\user} </username>
    <password> ${password} </password>
  </serverIdentity>
</authentication>
```

`proxy`

Optional.

Access the back-end application. 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 HTTP adapter's <authentication> element.

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>
```

`maxConcurrentConnectionsPerNode`

Optional.

Defines the maximum number of concurrent connections, which the MobileFirst Server can open to the back end.

IBM MobileFirst Platform Foundation for iOS does not limit the incoming service requests from applications. This subelement can be configured at the application server level. This product limits only the number of concurrent HTTP connections to the back-end service.

The default number of concurrent HTTP connections is 50. You can modify this number based on the expected concurrent requests to the adapter and the maximum requests allowed on the back-end service. You can also configure the back-end service to limit the number of concurrent incoming requests.

Consider a two-node system, where the expected load on the system is 100 concurrent requests and the back-end service can support up to 80 concurrent requests. You can set `maxConcurrentConnectionsPerNode` to 40. This setting ensures that no more than 80 concurrent requests are made to the back-end service.

If you increase the value, the back-end application needs more memory. To avoid memory issues, do not to set this value too high. Instead, estimate the average and peak number of transactions per second, and evaluate their average duration. Then, calculate the number of required concurrent connections as indicated in this example, and add a 5-10% margin. Then, monitor your back end, and adjust this value as required, to ensure that your back-end application can process all incoming requests.

When you deploy 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 the Scalability and Hardware Sizing document and the hardware calculator spreadsheet at Developer Center website for IBM MobileFirst Platform Foundation.

`connectionTimeoutInMilliseconds`

Optional.

The timeout in milliseconds until a connection to the back-end can be established. Setting this timeout does not ensure that a timeout exception occurs after a specific time elapses after the invocation of the HTTP request.

If you pass a different value for this parameter in the `invokeHTTP()` JavaScript function, you can override the value that is defined here. For more information, see the WL.Server class.

`socketTimeoutInMilliseconds`

Optional.

The timeout in milliseconds between two consecutive packets, starting from the connection packet. Setting this timeout does not ensure that a timeout exception occurs after a specific time elapses after the invocation of the HTTP request.

If you pass a different value for the **`socketTimeoutInMilliseconds`** parameter in the `invokeHttp()` JavaScript function, you can override the value that is defined here. For more information, see the WL.Server class.

JMS adapter **connectionPolicy** element:

You can use Java™ Messaging Service (JMS) is the standard messaging Java API to send messages between two or more clients. The MobileFirst JMS adapter provides reading and writing capabilities to messaging providers that implement the JMS API.

Structure

The connectionPolicy element has the following structure.

```
<connectionPolicy xsi:type="jms:JMSConnectionPolicyType">
  <!-- uncomment this if you want to use an external JNDI  repository -->
  <!-- <namingConnection url="MY_JNDI_URL"
  initialContextFactory="providers_intial_context_factory_class_name"
  user="JNDIUserName"
  password="JNDIPassword"/>
  -->

  <jmsConnection
    connectionFactory="java:comp/env/MYConnectionFactory"
    user="MessagingSystemUserName"
    password="MessagingSystemPassword"
  />
</connectionPolicy>
```

Attributes

The connectionPolicy element has the following attributes.

xsi:type

Mandatory.

The value of this attribute must be set to jms:JMSConnectionPolicyType.

Subelements

The connectionPolicy element has the following subelements.

namingConnection

Optional.

Describes how to connect to an external JNDI repository. Only used if the JNDI objects are not stored in the Java Platform, Enterprise Edition server that the adapter is deployed in. This element has the following attributes.

url

Mandatory.

The URL of the external JNDI repository. For example: iiop://*localhost*. The URL syntax is dependent upon 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, class loading problems 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.

User name of a user with authority to connect to the JNDI repository. If `user` is not specified, the default password is `guest`.

`jmsConnection`

Mandatory.

Describes the connection factory and optional security details that are used to connect to the messaging system.

`connectionFactory`

Mandatory.

The name of the connection factory that is used when you connect to the messaging system.

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 who is specified in the `user` attribute.

SAP Gateway adapter `connectionPolicy` element:

You can use SAP adapters to communicate with SAP Netweaver Gateway back-end services.

Structure

Note: Service Discovery now generates HTTP adapters for SAP Gateway connections instead of a specific SAP-type adapter.

The `connectionPolicy` element has the following structure.

```
<connectionPolicy xsi:type="nwgateway:NWGatewayHTTPConnectionPolicyType">
  <protocol>HTTP</protocol>
  <domain>host</domain>
  <port>8320</port>
  <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
  <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
  <serviceRootUrl>/sap/opu/odata/IWBEP/RMTSAMPLEFLIGHT_2/</serviceRootUrl>
  <authentication>
    <basic/>
    <serverIdentity>
      <client>001</client>
      <username>mygatewayuser</username>
      <password>mygatewaypassword</password>
    </serverIdentity>
  </authentication>
  <!-- Following properties used by adapter's key manager for choosing specific certificate from key store
  <sslCertificateAlias></sslCertificateAlias>
  <sslCertificatePassword></sslCertificatePassword>-->
  <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
</connectionPolicy>
```

Attributes

The `connectionPolicy` element has the following attributes.

`xsi:type`

Mandatory.

The value of this attribute must be set to `nwgateway:NWGatewayHTTPConnectionPolicyType`.

Subelements

The `connectionPolicy` element has the following subelements.

`protocol`

Mandatory.

The URL protocol to use. The following values are valid.

- `http`: default
- `https`

`domain`

Mandatory.

The host address.

`port`

Mandatory.

The port address.

`connectionTimeoutInMilliseconds`

Optional.

The time until a connection to the back-end can be established. Setting this timeout does not ensure that a timeout exception occurs after a time elapses after the invocation of the HTTP request.

If you pass a different value for this parameter in the `invokeHTTP()` JavaScript function, you can override this value. For more information, see the WL.Server class.

socketTimeoutInMilliseconds

Optional.

The time between two consecutive packets, starting from the connection packet. Setting this timeout does not ensure that a timeout exception occurs after a time elapses after the invocation of the HTTP request.

If you pass a different value for this parameter in the `invokeHttp()` JavaScript function, you can override the value that is defined here. For more information, see the WL.Server class.

serviceRootUrl

Mandatory.

The root URL for the SAP Netweaver gateway service that you are trying to access.

authentication

Mandatory.

Authentication configuration of the SAP Netweaver Gateway adapter. Your adapter can use one of four protocols, and a server identity, which you can configure if you define this element. You can define the `authentication` in one of the following ways.

- Basic Authentication

```
<authentication>
  <basic/>
</authentication>
```

- Digest Authentication

```
<authentication>
  <digest/>
</authentication>
```

- NTLM Authentication

```
<authentication>
  <ntlm workstation="value"/>
</authentication>
```

  The `workstation` attribute is optional. This attribute denotes the name of the computer on which the MobileFirst Server runs.

- SPNEGO/Kerberos Authentication

```
<authentication>
  <spnego stripPortOffServiceName="true"/>
</authentication>
```

  The attribute `stripPortOffServiceName` is optional. This attribute specifies whether the Kerberos client uses the service name without the port number.

  When you use this option, you must also place the `krb5.conf` file under *Project Name*`/server/conf`. This 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.

If the adapter exposes procedures with the attribute `connectAs="server"`, the connection policy can contain the `serverIdentity` element. This feature applies to all authentication schemes. For example:

```
<authentication>
  <basic/>
  <serverIdentity>
    <client>001</client>
    <username> ${DOMAIN\user} </username>
    <password> ${password} </password>
  </serverIdentity>
</authentication>
```

The `serverIdentity` element has the following 3 subelements.

`client`

Mandatory.

The SAP-Client to be used to contact the Netweaver Gateway.

`username`

Mandatory.

The user name for contacting the Netweaver Gateway.

`password`

Mandatory.

The password for contacting the Netweaver Gateway.

`sslCertificateAlias`

Optional for regular HTTP authentication and simple SSL authentication.

Mandatory for mutual SSL authentication.

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. For more information about the keystore setup process, see "SSL certificate keystore setup" on page 11-59.

`sslCertificatePassword`

Optional for regular HTTP authentication and simple SSL authentication.

Mandatory for mutual SSL authentication.

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. For more information about the keystore setup process, see "SSL certificate keystore setup" on page 11-59.

SAP JCo adapter `connectionPolicy` element:

You can use the SAP Java Connector (SAP JCo) adapters to develop SAP-compatible components and applications in Java.

Structure

The `connectionPolicy` element has the following structure.

```
<connectionPolicy xsi:type="sapjco:JCOConnectionPolicy"
  jcoClientClient="001"
  jcoClientUser="userName"
  jcoClientPasswd="password"
  jcoClientLang="EN"
  jcoClientAsHost="mysapserver.ibm.com"
  jcoClientSysnr="01"
  maxConnections="10"
/>
```

Attributes

The `connectionPolicy` element has the following attributes.

`xsi:type`

Mandatory.

The value of this attribute must be set to `sapjco:JCOConnectionPolicy`.

`jcoClientClient`

Mandatory.

The SAP client.

`jcoClientUser`

Mandatory.

The logon user.

`jcoClientPasswd`

Mandatory.

The logon password.

`jcoClientLang`

Mandatory.

The logon language.

`jcoClientAsHost`

Mandatory.

The application server.

`jcoClientSysnr`

Mandatory.

The SAP system number.

`maxConnections`

Mandatory.

The maximum number of connections that are allowed for the pool.

SQL adapter `connectionPolicy` element:

You can use the MobileFirst SQL adapter to run SQL queries with parameters and stored procedures that retrieve or update data in the database.

Structure

The `connectionPolicy` element has two options for connection:

- The `dataSourceDefinition` subelement: for development mode.
- The `dataSourceJNDIName` subelement: for production mode.

Connecting by using the `dataSourceDefinition` subelement:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceDefinition>
    <driverClass>com.mysql.jdbc.Driver</driverClass>
    <url>jdbc:mysql://localhost:3306/mydb</url>
    <user>myUsername</user>
    <password>myPassword</password>
  </dataSourceDefinition>
</connectionPolicy>
```

Connecting by using the `dataSourceJNDIName` subelement:

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceJNDIName>jdbc/myAdapterDS</dataSourceJNDIName>
</connectionPolicy>
```

Attributes

The `connectionPolicy` element has the following attributes.

`xsi:type`

Mandatory.

The value of this attribute must be set to `sql:SQLConnectionPolicy`.

Subelements

The `connectionPolicy` element contains either of the following subelements.

`dataSourceDefinition`

Mandatory

Contains the parameters that are needed to connect to a data source. The `url`, `user`, `password`, and `driverClass` parameters can be externalized as custom properties. Then, they can then be overridden by environment entries.

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=user_name
my-mysql-password=password
```

`dataSourceJNDIName`

Mandatory.

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 11-20.

You can also externalize the data source JNDI name and make it configurable from the server configuration. For example:

adapter.xml

```
<connectionPolicy xsi:type="sql:SQLConnectionPolicy">
  <dataSourceJNDIName>${my-adapter-ds}</dataSourceJNDIName>
</connectionPolicy>
```

worklight.properties

```
my - adapter - ds = jdbc / myAdapterDS
```

For more information, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

Developing JavaScript adapters

Learn to configure a new IBM MobileFirst Platform Foundation for iOS JavaScript adapter, and set up connectivity to back-end configuration.

Creating a JavaScript MobileFirst adapter:

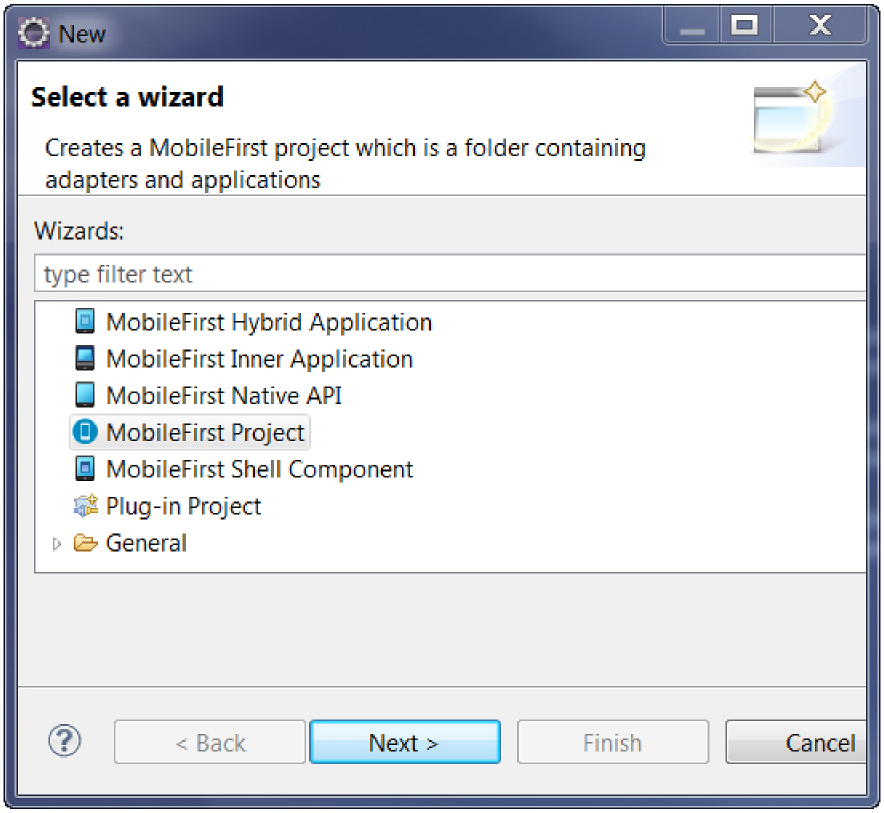
Follow these instructions to create a MobileFirst project and configure a new MobileFirst adapter.

About this task

On initial creation of a new adapter, MobileFirst 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 MobileFirst project. If you set up MobileFirst shortcuts, right-click anywhere within the Eclipse Project Explorer view and click **New** > **MobileFirst Project**. Otherwise, click **New** > **Other**, then select **MobileFirst** > **MobileFirst Project** from the list of wizards and click **Next**.



New

Select a wizard

Creates a MobileFirst project which is a folder containing adapters and applications

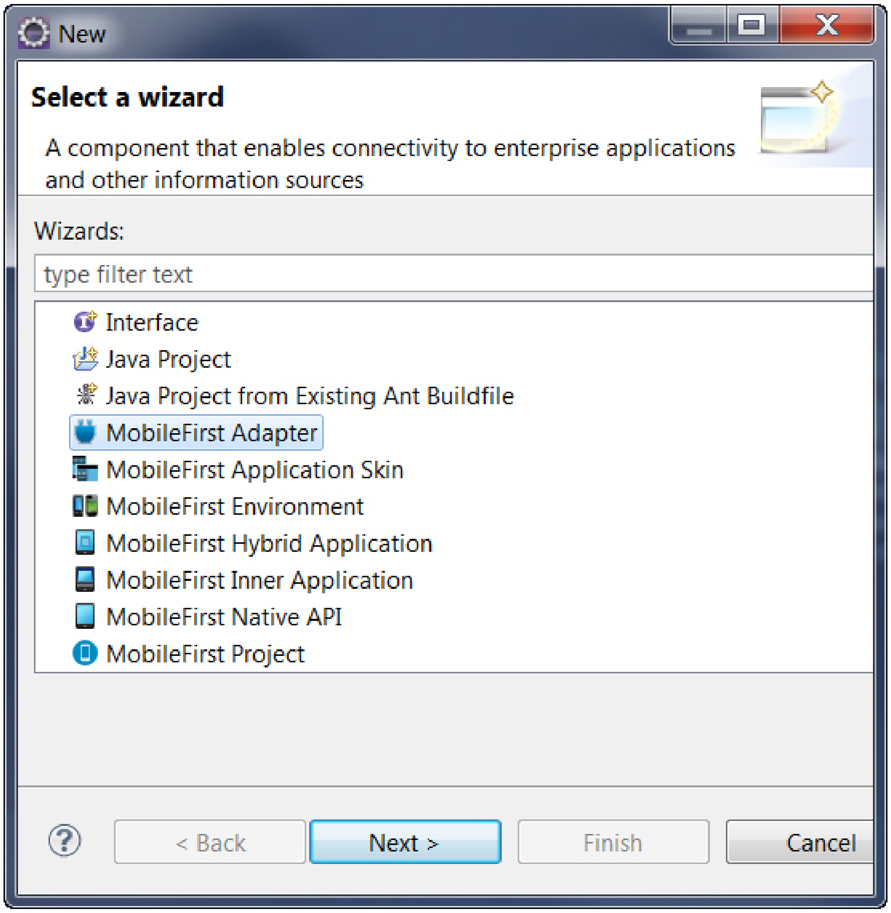
Wizards:

type filter text

- MobileFirst Hybrid Application
- MobileFirst Inner Application
- MobileFirst Native API
- MobileFirst Project
- MobileFirst Shell Component
- Plug-in Project
- General

Figure 8-8. Creating a MobileFirst project from the wizard.

2. In the **New MobileFirst Project** wizard, specify a name for the project and click **Finish**.
3. If you set up MobileFirst shortcuts, right-click the MobileFirst Project to which you want to add the adapter, and select **New** > **Adapter**. Otherwise, select **New** > **Other**, then select **MobileFirst** > **MobileFirst Adapter** from the list of wizards and click **Next**.



New

Select a wizard

A component that enables connectivity to enterprise applications and other information sources

Wizards:

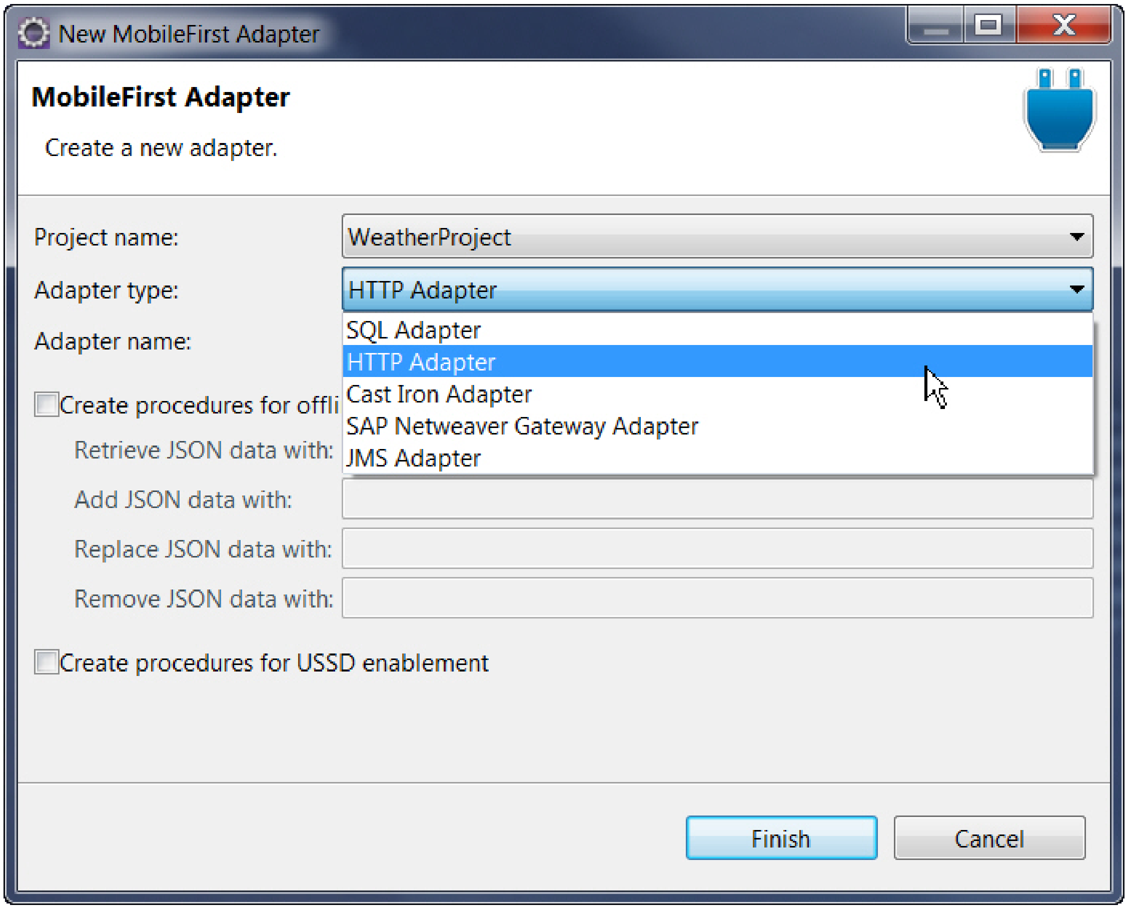
type filter text

- Interface
- Java Project
- Java Project from Existing Ant Buildfile
- MobileFirst Adapter
- MobileFirst Application Skin
- MobileFirst Environment
- MobileFirst Hybrid Application
- MobileFirst Inner Application
- MobileFirst Native API
- MobileFirst Project

Figure 8-9. Configuring a new MobileFirst adapter.

The **New Adapter** wizard opens.

4. Select the required adapter type from the **Adapter type** list and enter a name for the adapter in the **Adapter name** field.



New MobileFirst Adapter

MobileFirst Adapter

Create a new adapter.

Project name: WeatherProject

Adapter type: HTTP Adapter

Adapter name:

□ Create procedures for offli

Retrieve JSON data with:

Add JSON data with:

Replace JSON data with:

Remove JSON data with:

□ Create procedures for USSD enablement

Figure 8-10. 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 MobileFirst project, and generate the adapters that connect to those services.

The services discovery wizard supports the following types of back-end services:
- IBM Business Process Manager (IBM BPM) process applications. Discover IBM BPM process applications and generate the generic and application-specific adapters.
- Microsoft Azure Marketplace DataMarket.
- OData RESTful APIs.

- RESTful resources. Describe a RESTful resource by accessing your back-end service.
- 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.
- 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.

The adapters that communicate with the chosen back-end service are automatically generated, and placed in the **adapters** folder of your project.

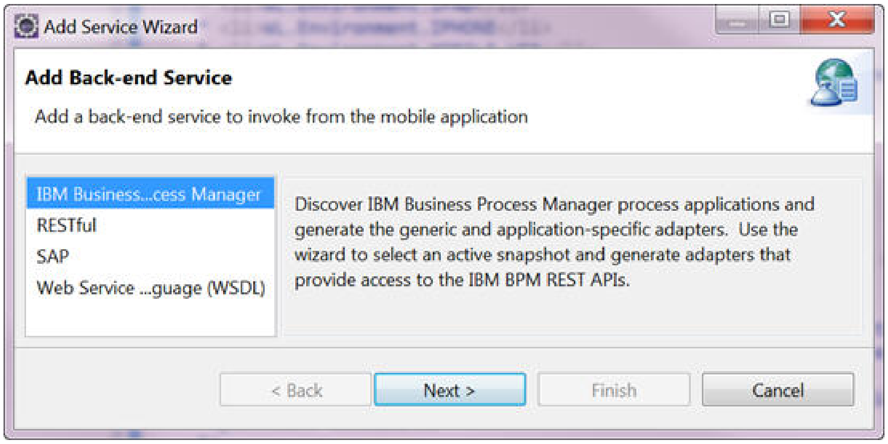
Note: If you manually modify an adapter file, first create a copy of this file, and make sure to modify only the copied file. The services discovery wizard might regenerate the original file each time you add a service. The exact adapter that is regenerated depends on the type of service that is involved.

Generating adapters with IBM BPM:

You can use the Add Service Wizard to select an active snapshot and generate an adapter that provides access to the IBM Business Process Manager (IBM BPM) REST APIs.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.



Add Service Wizard

Add Back-end Service

Add a back-end service to invoke from the mobile application

Discover IBM Business Process Manager process applications and generate the generic and application-specific adapters. Use the wizard to select an active snapshot and generate adapters that provide access to the IBM BPM REST APIs.

Figure 8-11. The Add Service Wizard

2. Select **IBM Business Process Manager** and click **Next**. The IBM BPM process applications page is displayed.
3. You can set up a connection to an IBM BPM server if you complete one of the following choices.
   - Click **Add** next to the **Connections** field.
   - Click **Manage IBM BPM Connections** to edit existing connections.
     a. Click **Add**.

- Select an existing IBM BPM connection from the drop-down list in the **Connections** field.

4. Enter your server URL, user, and password credentials and click **OK**. The IBM BPM process applications and their available active snapshots are visible in the **Select Snapshop** field.
5. Expand the process application that you want to create an adapter for, select a snapshot, and click **Next**.
6. Name your adapter and click **Finish**.

Results

A generic adapter and an application-specific adapter are generated under the **adapters** folder of your project. You can use this adapter to access IBM BPM REST APIs on the IBM BPM server.

The project's `/server/conf/worklight.properties` file is also updated to include the necessary IBM BPM server properties.

Generating adapters by using Microsoft Azure:

You can use the Add Service Wizard to generate adapters that discover your subscribed data services from Microsoft Azure Marketplace DataMarket.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.
2. Select **Microsoft Azure Marketplace DataMarket** and click **Next**.
3. Set up a connection to a server by either:
   - Clicking **Add** to create an Azure connection.
   - Clicking the **Manage Azure Connections** link to edit existing connections.
   - Selecting an existing Azure connection from the **Connection** list.
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. You can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

Generating adapters with OData:

You can use the Add Service Wizard to generate adapters that access the OData RESTful APIs, and explore the OData metadata for a service URL.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.
2. Select **OData** and click **Next**.
3. Set up a connection to a server by either:

- Clicking **Add** to create an OData connection.
- Clicking the **Manage OData Connections** link to edit existing connections.
- Selecting an existing OData connection from the **Connection** list.

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.

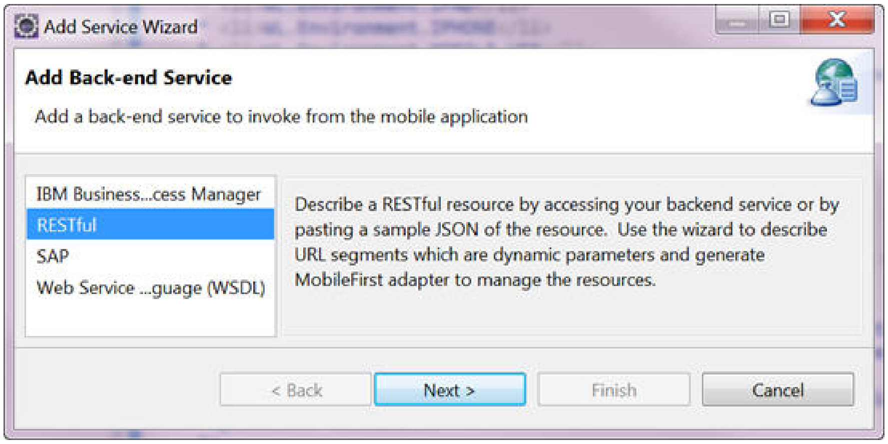
- An `.xml` service description file is also generated under the **services** folder of your project. You can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

Generating adapters by describing RESTful resources:

You can use the Add Service wizard to describe a RESTful resource to access your back-end service. Use the wizard to describe URL segments that are dynamic parameters to generate MobileFirst adapters.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.
2. Select **RESTful** and click **Next**.
   The **Resource Structure** page is visible.



Add Back-end Service

Add a back-end service to invoke from the mobile application

Describe a RESTful resource by accessing your backend service or by pasting a sample JSON of the resource. Use the wizard to describe URL segments which are dynamic parameters and generate MobileFirst adapter to manage the resources.

Figure 8-12. The Add Service Wizard

3. Complete the **Resource Structure** page.
   a. Type the URL to the resource into the **Resource URL** field and complete one of the following choices.
      - Click **Fetch URL** to retrieve a JSON payload from the URL entered and display it in the **Sample Resource (as JSON)** field.

        You might need to set up authentication and custom headers before you click **Fetch URL**.
      - Manually enter a sample JSON payload in the **Sample Resource (as JSON)** field.

The **Sample Resource (as JSON)** field is completed with the JSON code that results from the provided URL. That JSON code is parsed into build structures that are visible in the **Resource Structure** field.

b. From the **Resource Structure** field, select which parameter is the **Key**.

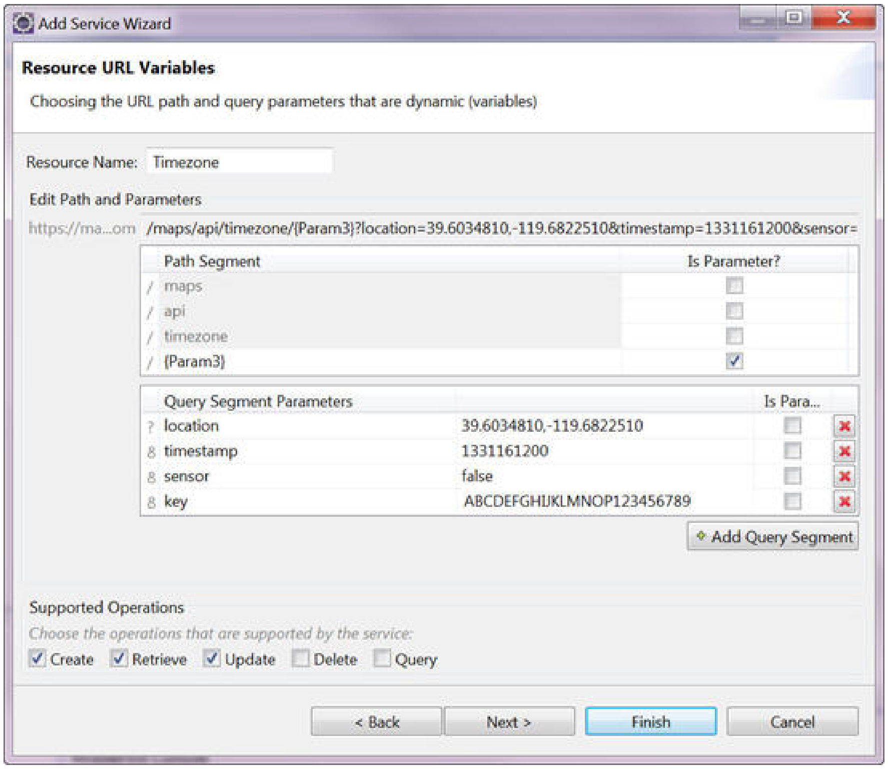
c. Select the level of authentication from the **Authentication** field.

- **None**: Authentication is not needed.
- **Basic Authentication**: Authentication is needed. The user ID and password are sent in clear text, according to HTTP Basic Authentication guidelines.
- **Digest**: Authentication is needed. The user ID and password are sent by using HTTP Digest authentication (as a hash).

d. If you need specific HTTP headers to communicate with the back-end service, complete the **Request Custom HTTP Header** fields. The left column is the header name, and the right column is the value. The drop-down list on the left includes a set of HTTP header names. Or, you can type any header name.

e. Click **Next**.

4. From the Resource URL Variables page, select the settings for your adapter.



Resource URL Variables

Choosing the URL path and query parameters that are dynamic (variables)

Resource Name: Timezone

Edit Path and Parameters

https://ma...om /maps/api/timezone/{Param3}?location=39.6034810,-119.6822510&timestamp=1331161200&sensor=

	Path Segment	Is Parameter?
/	maps	□
/	api	□
/	timezone	□
/	{Param3}	☑

	Query Segment Parameters		Is Para...	
?	location	39.6034810,-119.6822510	□	☒
&	timestamp	1331161200	□	☒
&	sensor	false	□	☒
&	key	ABCDEFGHIJKLMNOP123456789	□	☒

Add Query Segment

Supported Operations

Choose the operations that are supported by the service:

☑ Create ☑ Retrieve ☑ Update □ Delete □ Query

5. To customize resource operations, click **Next**. Otherwise, click **Finish**.

6. If you clicked **Next**, customize your resource operations.

a. Choose the **HTTP Method** for each operation.

Table 8-3. HTTP methods

Method	Safe	Idempotent	Cacheable	Description
GET	Yes	Yes	Yes	Retrieves information from the input server by using the input URI
PUT	No	Yes	No	Puts or updates a resource on the server
POST	No	No	No	Sends data to the server
DELETE	No	Yes	No	Deletes the requested resource

b. Complete the **Edit path and parameters** field.

c. If you need specific HTTP headers to communicate with the back-end service, complete the **Request Custom HTTP Header** fields. The left column is the header name, and the right column is the value. The drop-down list on the left includes a set of HTTP header names. Or, you can type any header name.

d. Click **Finish**.

Results

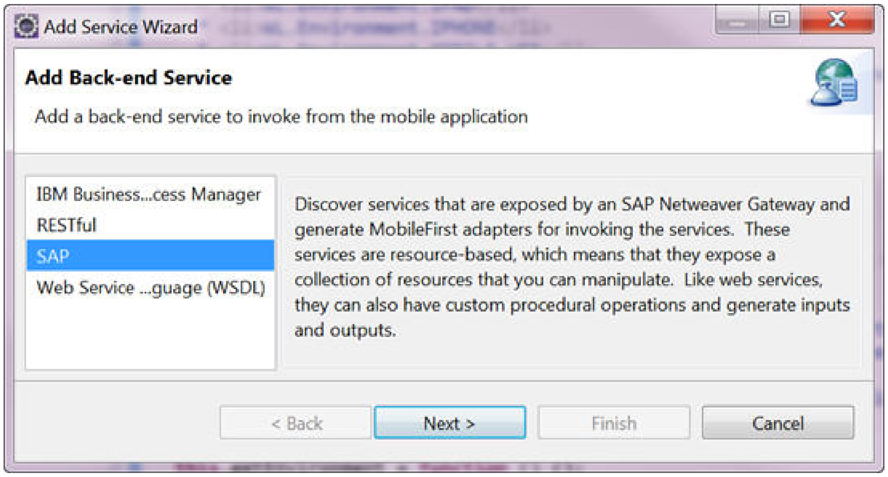
- An adapter is generated under the **adapters** folder of your project. You can use this adapter to invoke services with JavaScript calls.
- A `.xml` service description file is also generated under the **services** folder of your project. You can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

Generating adapters with SAP Gateway:

You can use the Add Service Wizard to discover services that are exposed by an SAP Gateway and generate MobileFirst adapter for invoking the services.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.


Add Service Wizard
Add Back-end Service
Add a back-end service to invoke from the mobile application
IBM Business...cess Manager
RESTful
SAP
Web Service ...guage (WSDL)
< Back
Next >
Finish
Cancel

Discover services that are exposed by an SAP Netweaver Gateway and generate MobileFirst adapters for invoking the services. 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.

Figure 8-13. The Add Service Wizard

2. Select **SAP** and click **Next**.
3. 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** list.
4. 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.
5. Click **Finish**.

Results

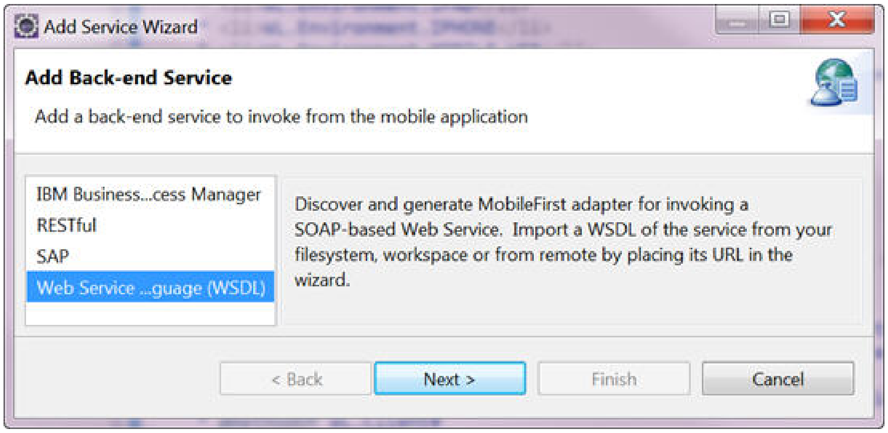
- An adapter is generated under the **adapters** folder of your project. You can use this adapter to invoke services with JavaScript calls.
- An `.xml` service description file is also generated under the **services** folder of your project. You can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

Generating adapters with WSDL:

You can use the Add Service Wizard to discover and generate MobileFirst adapters for invoking a SOAP-based Web Service. You can import a Web Service Definition Language (WSDL) of the service from your file system, workspace, or remotely by typing its URL in the wizard.

Procedure

1. Right-click the **services** folder of your project in the Project Explorer tab, and click **Discover Back-end Services**. The Add Service Wizard window opens.



Add Service Wizard

Add Back-end Service

Add a back-end service to invoke from the mobile application

IBM Business...cess Manager

RESTful

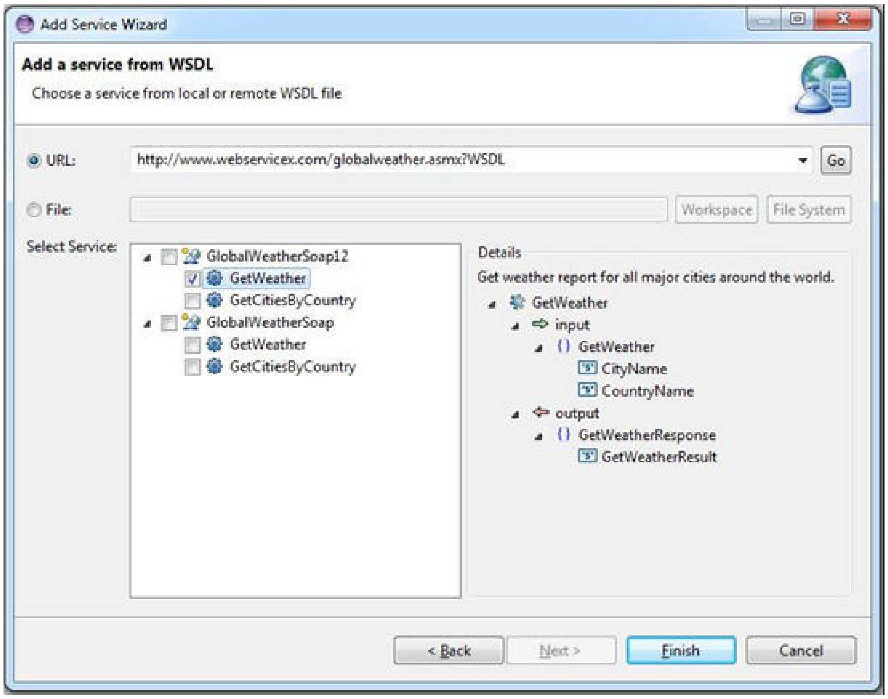
SAP

Web Service ...guage (WSDL)

Discover and generate MobileFirst adapter for invoking a SOAP-based Web Service. Import a WSDL of the service from your filesystem, workspace or from remote by placing its URL in the wizard.

Figure 8-14. The Add Service Wizard

2. Select **Web Service Definition Language (WSDL)** and click **Next**.
3. Complete the Add a Service from WSDL page.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.



Add Service Wizard

Add a service from WSDL

Choose a service from local or remote WSDL file

● URL: http://www.webservicex.com/globalweather.asmx?WSDL

○ File:

Select Service:

- □ GlobalWeatherSoap12
  - ☑ GetWeather
  - □ GetCitiesByCountry
- □ GlobalWeatherSoap
  - □ GetWeather
  - □ GetCitiesByCountry

Details

Get weather report for all major cities around the world.

- GetWeather
  - input
    - GetWeather
      - CityName
      - CountryName
  - output
    - GetWeatherResponse
      - GetWeatherResult

a. Enter a URL or select one from the **URL** drop-down list, and click **Go**.

   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 one or more services that you want to invoke from your application.

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. You can refer to the `.xml` files under the **services** folder of your project to have a summary view of the target adapters.

  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. 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.

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 MobileFirst application in the same way as you invoke other MobileFirst 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 MobileFirst Platform Foundation for iOS 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' }
```

Setting up connectivity to back-end configuration:

Learn about adapter timeout and concurrency, connectivity for non-Java adapters, and developing JavaScript adapter code.

Adapter timeout and concurrency:

Changes in the behavior of adapter timeout and concurrency starting in IBM MobileFirst Platform Foundation for iOS V6.3 have impact on the adapter XML schema. Adapter failures can occur if the JSON data exceeds available memory.

Timeout and concurrency in IBM MobileFirst Platform Foundation for iOS V6.3

Starting in IBM MobileFirst Platform Foundation for iOS V6.3, the behavior of timeout and concurrency was modified for HTTP-based, JMS, and SQL adapters.

HTTP-based adapters

In earlier versions of IBM MobileFirst Platform Foundation for iOS, timeout and concurrency were handled by thread pools. They are now enforced by the underlying HTTP framework.

- Timeout:

Previously, you were able to define a timeout for a single procedure. Starting from V6.3, you can use the standard socket timeout and connection timeout that are provided by HTTP frameworks. A socket timeout defines the time between two consecutive packets, starting from the connection packet. A connection timeout defines the time within which a connection to the back-end must be established.

You can set the socket and connection timeouts in two places:

  - To set the default value for an adapter, set it in the adapter XML file. For more details on timeout in the XML file, see "Structure of the adapter XML file" on page 8-38.
  - To set the timeouts for a specific back-end invocation, use the **`options`** struct in the `invokeHttp()` JavaScript function. For more information, see the WL.Server class.

- Concurrency:

  You use the **`<maxConcurrentConnectionsPerNode>`** subelement of the **`<connectionPolicy>`** element in the adapter XML file to set concurrency limits. For more information, see "Structure of the adapter XML file" on page 8-38.

JMS adapters

- Timeout: JMS adapter provides several timeout specifications based on the action that is attempted. There has been no change in functionality in that sense but the per-procedure timeout has been removed.
- Concurrency: Starting from IBM MobileFirst Platform Foundation for iOS V6.3, concurrency is no longer supported for JMS adapters.

SQL adapters

Previously, concurrency and timeout were handled by thread pools. Starting from V6.3, the only method to define concurrency and timeout, is to use the SQL Connection Pool obtained using a JNDI reference.

XML schema changes in IBM MobileFirst Platform Foundation for iOS V6.3

IBM MobileFirst Platform Foundation for iOS V6.3 uses a new adapter XML schema. Adapters using earlier schemas cannot be used in Studio V6.3 and CLI V6.3. The following changes are made to the adapter schema during upgrade:

- The **`requestTimeoutInSeconds`** attribute of the **`<procedure>`** element is no longer supported. During project upgrades to IBM MobileFirst Platform Foundation for iOS V6.3, the attribute is commented out in all **`<procedure>`** elements.
- The **`<loadConstraints>`** element and its attributes are no longer supported. During project upgrades to IBM MobileFirst Platform Foundation for iOS V6.3, this element is removed.
- In HTTP-based adapters, there are three new elements: **`<connectionTimeoutInMilliseconds>`**, **`<socketTimeoutInMilliseconds>`**, and **`<maxConcurrentConnectionsPerNode>`** under **`<connectionPolicy>`**. During project upgrades to IBM MobileFirst Platform Foundation for iOS V6.3, these new elements are added. For more information, see "Structure of the adapter XML file" on page 8-38.

Compatibility with earlier versions

Adapters from previous versions work and can be deployed on the MobileFirst Server V6.3. However, they behave differently.

HTTP-based adapters

- Timeout:

  The value of the **requestTimeoutInSeconds** attribute of **<procedure>** elements is now used to set the value of the HTTP socket timeout and connection timeout per procedure.
- Concurrency:

  The concurrency limits are enforced by the underlying HTTP framework, instead of by a thread pool.

JMS adapters
The **requestTimeoutInSeconds** attribute of **<procedure>** and **<loadConstraints>** elements is ignored.

SQL adapters
The **requestTimeoutInSeconds** attribute of **<procedure>** and **<loadConstraints>** elements is ignored. Use JNDI configuration instead.

Adapter invocation failures due to large data

Adapter calls are not intended for returning very large JSON data. The adapter response is stored in memory and string parsed. Data that exceeds the amount of available memory might cause adapter invocation to fail. To reduce the possibility that such a failure occurs, limit the amount of data to less than 10 MB.

Connectivity for non-Java adapters:

With IBM MobileFirst Platform Studio, you can configure back-end connectivity for non-Java adapters.

HTTP adapters:

The MobileFirst 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 MobileFirst 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 11-59.
- 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 "HTTP adapter `connectionPolicy` element" on page 8-43.

- If you use WebSphere Application Server, you can benefit from the WebSphere SSL configuration as described in WebSphere Application Server SSL configuration and HTTP adapters.

Note, however that SSL represents transport level security, which is independent of basic authentication. It is possible to do basic authentication either over HTTP or HTTPS.

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 MobileFirst Platform Foundation for iOS 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>;
```

NTLM configuration for HTTP adapters:

NTLM uses a challenge-response mechanism for authentication. Learn how to use a MobileFirst adapter when connecting to a back end or resource that is protected by NTLM protocol.

Back-end connection types

MobileFirst Server can connect with a back-end system or resource that is protected by NTLM protocol, either as a server or as an end-user.

Connect as server
In this connection type, all sessions use the same connection context to the back end. This is the default MobileFirst Server behavior. See Figure 1.

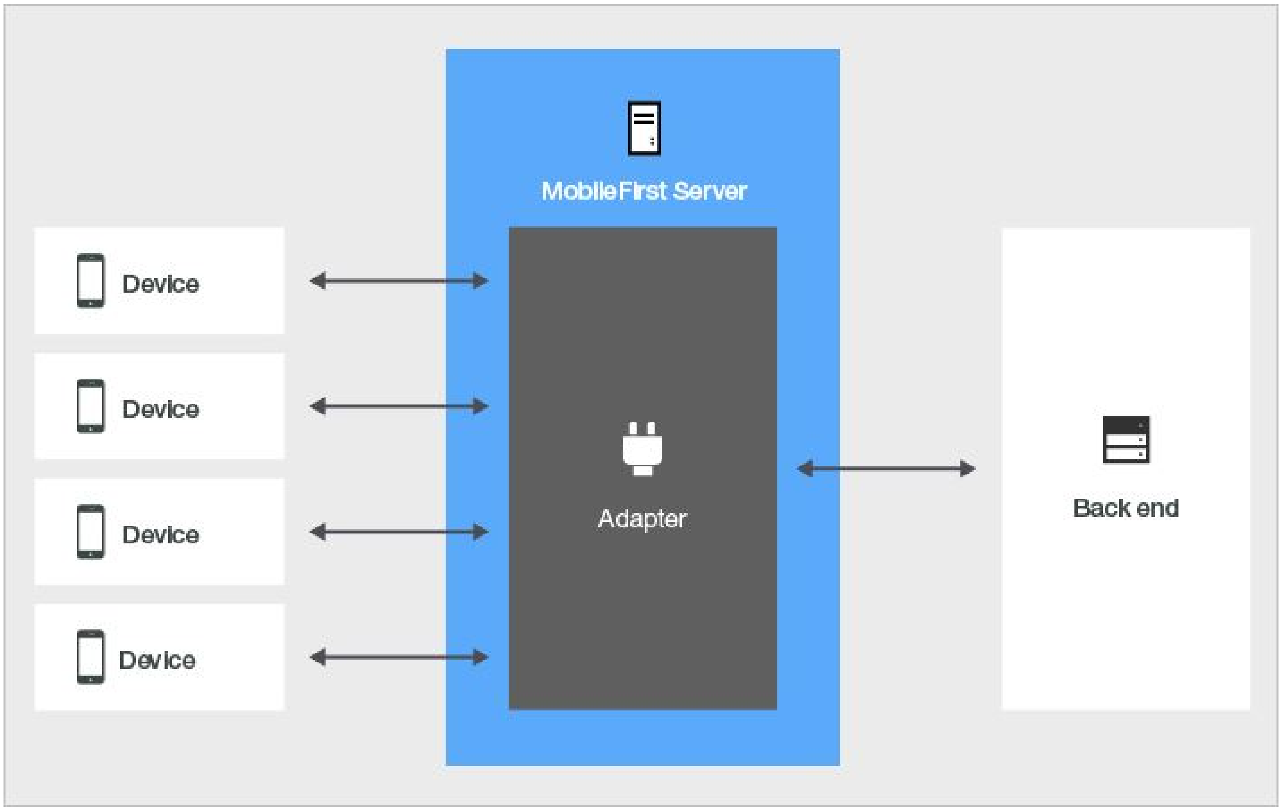


Figure 8-15. Connecting to the back end as server

Connect as end user
In this connection type, each session is authenticated separately and has a unique connection context against the back end. See Figure 2.

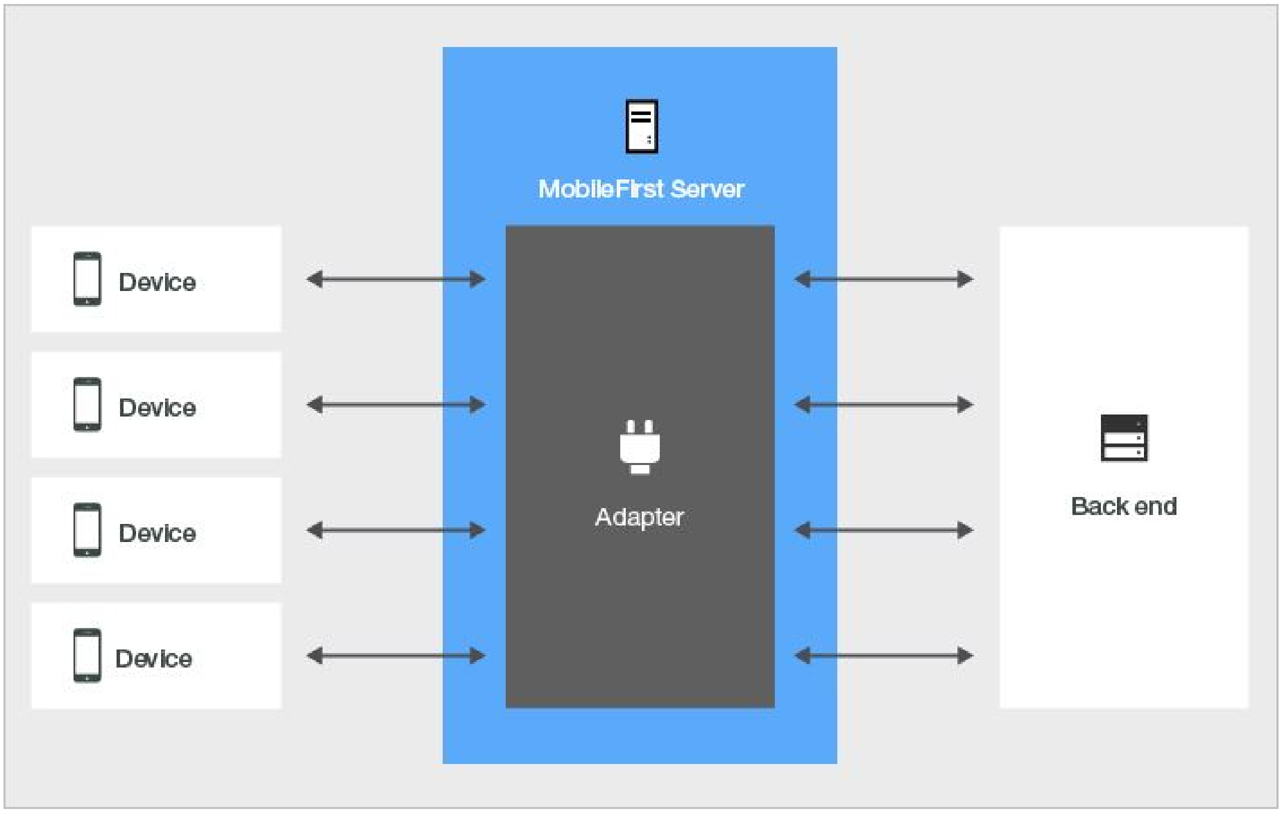


Figure 8-16. Connecting to the back end as end user

Back-end connection settings

The connection types that are described in the previous section correspond to the `server` and `endUser` options of the `connectAs` attribute in the adapter XML file.

- To connect as server (the default), there is no need to make any configuration change to the `adapter.xml` file.
- To connect as an end-user, add a `procedure` element to the adapter XML file, as follows:

```
<procedure name= "MyProcedure" connectAs= "endUser" />
```

For more information about the `connectAs` attribute, see "Structure of the adapter XML file" on page 8-38.

Implementing NTLM authentication for server requests:

Implement NTLM authentication where a single, shared instance of HTTP client is used per adapter for all mobile application instances.

About this task

By default, procedures that connect to a back-end server that uses NTLM protocol are handled on a server-to-server basis, as the `connectAs` attribute is defined as `="server"`. You need only configure `serverIdentity` of the adapter XML file as a subelement of the `authentication`. You also add the `ntlm workstation` attribute, so that MobileFirst Server knows which authentication method to use when connecting to the back end.

Procedure

1. Open the adapter.xml file of the project.
2. Add an authentication element as follows:

```
<connectivity>
    <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
        <protocol>http</protocol>
        <domain>your-domain-here</domain>
        <port>80</port>
        <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
        <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
        <authentication>
            <ntlm workstation="ServerName"/>
            <serverIdentity>
                <username>your-server-name-here/your-username-here</username>
                <password>your-password-here</password>
            </serverIdentity>
        </authentication>
        <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
    </connectionPolicy>
</connectivity>
```

Note: When the NTLM protocol is used, the user name must always be specified in the format: server-name/user-name. Ensure that in the adapter.xml file, you pass the server and user names to the back-end server in that pattern.

For more information, see "HTTP adapter connectionPolicy element" on page 8-43.

Implementing NTLM authentication for end-user requests:

Implement NTLM authentication where a separate instance of HTTP session is opened for each client session.

About this task

To enable NTLM authentication with connectAs="endUser", you configure the server and add handling functions in the adapter.

Configure MobileFirst Server authentication:

Procedure

1. In the authenticationConfig.xml file for your project, make the following edits:
   a. Create a security test to protect the procedure:

   ```
   <customSecurityTest name="NTLMSecurityTest">
       <test isInternalUserID="true" realm="NTLMAuthRealm"/>
   </customSecurityTest>
   ```

   b. Use BasicAuthenticator, AdapterBasedAuthenticator, or any other authenticator that handles userIdentity, as the class for the realm to be used by the security test:

   ```
   <realm name="NTLMAuthRealm" loginModule="AuthLoginModule">
       <className>com.worklight.integration.auth.AdapterAuthenticator</className>
       <parameter name="login-function" value="MyAdapter.onAuthRequired"/>
       <parameter name="logout-function" value="MyAdapter.onLogout"/>
   </realm>
   ```

   c. Add a login module to create and store user identities to be used by this realm:

   ```
   <loginModule name="AuthLoginModule">
       <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
   </loginModule>
   ```

2. In the adapter.xml file, make the following edits:
   a. Add the authentication element and its ntlm workstation subelement to the adapter XML file, so that MobileFirst Server knows which authentication method to use when connecting to the back end:

      ```
      <connectivity>
          <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
              <protocol>http</protocol>
              <domain>Put.Your.Domain.Here</domain>
              <port>80</port>
              <connectionTimeoutInMilliseconds>30000</connectionTimeoutInMilliseconds>
              <socketTimeoutInMilliseconds>30000</socketTimeoutInMilliseconds>
              <authentication>
                  <ntlm workstation="wl-ntlm"/>
              </authentication>
              <maxConcurrentConnectionsPerNode>50</maxConcurrentConnectionsPerNode>
          </connectionPolicy>
      </connectivity>
      ```

   b. Assign this security test to the procedure that is used to connect to the back end protected by the NTLM protocol, and add connectAs="endUser" to the procedure declaration in the adapter XML file:

      ```
      <procedure name= "getNTLMData" securityTest= "NTLMSecurityTest" connectAs= "endUser"/>
      ```

Create adapter functions:
Procedure

1. Define the submitAuthentication function. Create a userIdentity that contains a user identifier and credentials properties. Format the userId in the pattern servername/username:

   ```
   function submitAuthentication(username, password){
       var userIdentity = {
           userId: "MyServerName\/"+ username,
           credentials: password
       };
       WL.Server.setActiveUser("NTLMAuthRealm", null);
       WL.Server.setActiveUser("NTLMAuthRealm", userIdentity);
       ...
   }
   ```

   For more information, see “Implementing adapter-based authenticators” on page 8-284.
2. Create an HTTP request to the NTLM-protected back end:

   ```
   function getSecretData(){
       var input = {
           method : 'get',
           returnedContentType : 'html',
           path : "index.html"
       };
       return WL.Server.invokeHttp(input);
   }
   ```

   For more information, see the invokehttp() function in the WL.Server class.

Back-end 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 "MobileFirst JavaScript adapters" on page 8-36. The following sections explain how to handle back-end 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 WLClient class to invoke one of the adapter's procedures from the MobileFirst Server.
2. The adapter then uses the `invokeHttp` method of the WL.Server class to call the back-end service.
3. The adapter procedure processes the data from the back end 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 WLClient 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 8-4. 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 back-end service, only from the connection with the MobileFirst 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 MobileFirst Server is unresponsive (for example, due to a bad host name or because the MobileFirst 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 back-end service. This means, for example, that if your procedure calls a back-end 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 back-end service returns an error.

Invocations from the adapter to the back end

From your procedure, you call a remote back-end 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 back-end service.

Table 8-5. Properties of the object received following the invocation from the adapter to the back end

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.
statusReason	Short text description that explains the status code.
totalTime	Response time plus any additional time for IBM MobileFirst Platform Foundation for iOS 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 whereasIBM MobileFirst Platform Foundation for iOS 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 back-end response might include additional data that should not be forwarded to the client; such as certain response headers, architecture of the back end, 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 using the **`mfp add adapter`** command. The back end host name is set to `www.ibm.com`.

```
<?xml version="1.0" encoding="UTF-8"?>
<wl:adapter name="adapt"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wl="http://www.ibm.com/mfp/integration"
  xmlns:http="http://www.ibm.com/mfp/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>
    </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`.

```
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;
}
```

SQL adapters:

You can use the IBM MobileFirst Platform Foundation for iOS SQL adapter to execute parameterized SQL queries and stored procedures that retrieve or update data in an SQL database.

You can use plain SQL queries or stored procedures. As a developer, you must download the JDBC connector driver for the specific database type separately and add it to the `server\lib\` folder of a MobileFirst project. You can download the JDBC connector driver from the appropriate vendor website.

For more information about SQL adapters, see the JavaScript SQL Adapter tutorial on the Getting Started of the Developer Center.

JMS adapters:

Java messaging service (JMS) is the standard messaging Java API for sending messages between two or more clients. The MobileFirst JMS adapter provides reading and writing capabilities to messaging providers that implement the JMS API.

You can configure a JMS adapter to work with such messaging providers as a Liberty profile server or a WebSphere MQ message broker.

Connecting a JMS adapter to the WebSphere Application Server messaging provider:

You can develop and test MobileFirst adapters that use Java Message Service (JMS) on a WebSphere Application Server messaging provider. The WebSphere Application Server messaging provider can be the default messaging provider, a WebSphere MQ messaging provider, or a third-party provider.

Before you begin

You must have configured the WebSphere Application Server messaging provider and JMS resources such as the queue connection factories and the queues or topics.

About this task

The following procedure shows how to connect a JMS adapter to a WebSphere Application Server messaging provider.

Procedure

1. Create a MobileFirst JMS adapter.
2. Because the adapter runs on a JMS-enabled server, the naming connection section of the `adapter.xml` file is not necessary. It can remain commented out.
3. Enter the JNDI name for the queue 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/WASQCF"
      user="admin"
      password="admin"
      />
  </connectionPolicy>

  <loadConstraints maxConcurrentConnectionsPerNode="10"/>
</connectivity>
```

4. 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/WASQueue",
    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;
  };
}
```

5. Change the MobileFirst target server in MobileFirst Platform Command Line Interface for iOS to point to your WebSphere Application Server environment.
6. Build and deploy the MobileFirst adapter to the WebSphere Application Server environment. You can test the JMS adapter in your browser by using the following URL syntax:

```
http://<was-hostname>:<port>/<context-root>/invoke?adapterName=
<adapterName>&procedure=<procedureName>&parameters=["<parameters>"]
```

An example of a URL pointing to an external WebSphere Application Server server:

```
http://localhost:9080/worklight/invoke?adapter=JMSAdapter&procedure=
writeMessage&parameters=["Hello World"]
```

Connecting a JMS adapter to a Liberty profile server:

You can develop and test MobileFirst 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 IBM MobileFirst Platform Foundation for iOS does not contain the built-in Liberty JMS features. Therefore, an embedded MobileFirst Development Server or a local external instance of this bundled WebSphere Application Server 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 your MobileFirst tools, 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 MobileFirst 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 WebSphere Application Server 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, as 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 MobileFirst 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>
   </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 MobileFirst target server in MobileFirst Platform Command Line Interface for iOS to point to your Liberty ND server.
7. Build and deploy the MobileFirst 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 JMS adapter to WebSphere MQ:

You can connect a MobileFirst Java Message Service (JMS) adapter to WebSphere MQ.

Before you begin

If you are running the adapter on WebSphere Application Server, use the WebSphere Application Server messaging provider. For more information, see "Connecting a JMS adapter to the WebSphere Application Server messaging provider" on page 8-78.

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 MobileFirst JMS adapter does not support connecting to WebSphere MQ through bindings mode, only in client mode. A TCP connection is created for each JMS request, even if the JMS broker and MobileFirst adapter are running on the same computer.

To connect a MobileFirst JMS adapter to WebSphere MQ, you create a project, copy some JAR files to the project directory, and modify the adapter file.

Procedure

Include the required WebSphere MQ Java libraries

1. Create a MobileFirst 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 MobileFirst 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 MobileFirst 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 security 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>
   </connectivity>
   ```

Modify the adapter implementation file

10. Open the adapter's implementation file.
11. In the autogenerated 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 MobileFirst 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']
```

Microsoft Azure OData adapters:

Your IBM MobileFirst Platform Foundation for iOS applications can communicate with back-end services by using Microsoft Azure OData adapters. Using HTTP rest calls and the OData protocol, applications can remotely retrieve and query entities through the adapter.

You can use Microsoft Azure Odata adapters to retrieve and query entities that exist on a back-end system. For more information, see "Retrieving an entity" on page 8-87 and "Querying an existing entity" on page 8-90.

OData adapters:

Your IBM MobileFirst Platform Foundation for iOS applications can communicate with back-end services by using OData adapters. Using HTTP rest calls and the OData protocol, applications can remotely create, retrieve, update, and delete entities through the adapter.

You can use OData adapters to create, retrieve, update, delete, and analyze Entities that exist on a back-end system. For more information about invoking an adapter, see "Starting an adapter" on page 8-86.

SAP Gateway adapters:

Your IBM MobileFirst Platform Foundation for iOS applications can communicate with back-end services by using an SAP Gateway adapter. Using HTTP rest calls and the OData protocol, applications can remotely create, retrieve, update, and delete entities through the adapter.

Note: Starting with V7.1.0, communication to an SAP Gateway is done by using regular HTTP adapters.

For information about invoking an adapter, see "Starting an adapter" on page 8-86.

Configuring an SAP Gateway adapter with a system user:

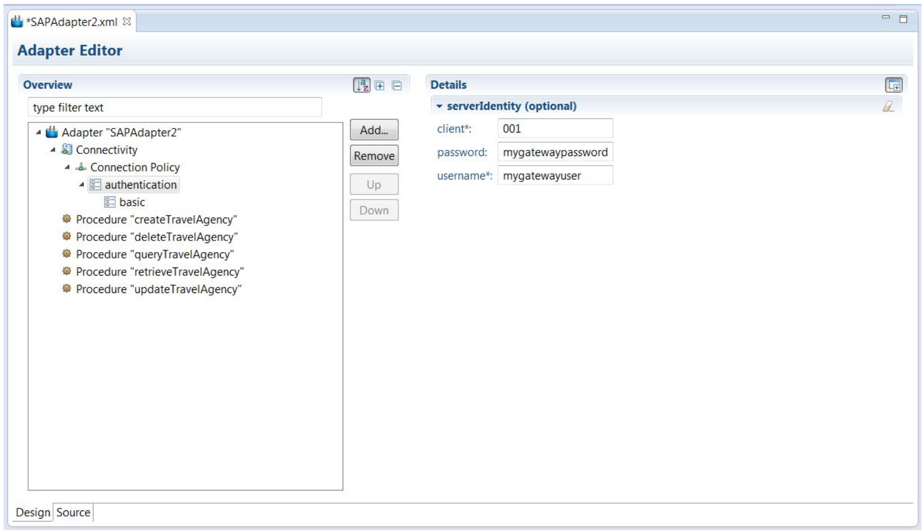
You can connect to an SAP back end with a system user. However, you should use user-based authentication for most scenarios. Check your SAP license terms.

Before you begin

Make sure that SAP Gateway adapters exist in yourMobileFirst project.

Procedure

1. Expand the **adapters** folder of your MobileFirst project, right-click **<*SAP Adapter name*>.xml**, and select **Open With** > **Adapter Editor**.
2. In the **Adapter Editor** view, expand **Connectivity** and select **Connection Policy**.
3. Click **Add**, select **authentication** from the Add Item window, and click **OK**.
4. Under Connection Policy, select **authentication** and click **Add**.
   a. Select the appropriate authentication mechanism of your SAP server. This example uses **basic**. The results look something like the following image.


*SAPAdapter2.xml
Adapter Editor
Overview
type filter text
Adapter "SAPAdapter2"
Connectivity
Connection Policy
authentication
basic
Procedure "createTravelAgency"
Procedure "deleteTravelAgency"
Procedure "queryTravelAgency"
Procedure "retrieveTravelAgency"
Procedure "updateTravelAgency"
Add...
Remove
Up
Down
Details
serverIdentity (optional)
Design
Source

client*: 001
password: mygatewaypassword
username*: mygatewayuser

5. To see the changes in the source code, select the **Source** tab at the bottom of the **Adapter Editor** view.

Results

Your SAP Gateway adapter is configured to start procedures on a server-identity authentication basis.

SAP JCo adapters:

You can use the SAP Java Connector (SAP JCo) adapters to develop SAP-compatible components and applications in Java.

Invoking an SAP JCo adapter:

If you invoke the SAP Java Connector (SAP JCo) adapter, you can communicate with the SAP Server both inbound and outbound.

About this task

Table 8-6. Parameters

Parameter	Mandatory or Optional	Description
`FunctionName`	Mandatory	The name of the function to be run on the SAP system
`Imports`	Mandatory	The input parameters

Procedure

1. Call the `invokeSAPFunction` procedure and pass in the following parameters. **`{"FunctionName" : "BAPI_USER_GET_DETAIL", "Imports" : {"USERNAME" : "DEVELOPER"}}`**
2. Save your *`adapter_name`*`-impls.js` file.

Results

The following message is displayed in the Console: `Adapter build and deploy finished.`

The Cast Iron adapter:

The MobileFirst 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 IBM MobileFirst Platform Foundation for iOS.

Inbound pattern.
Cast Iron sends notifications to devices through IBM MobileFirst Platform Foundation for iOS.

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 MobileFirst 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.

Troubleshooting a Cast Iron adapter – connectivity issues

Symptom: The MobileFirst 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.

Starting an adapter:

You can create, retrieve, update, delete, and analyze `Entities` that exist on an back-end system by using a MobileFirst adapter.

Creating an entity:

You can create entity remotely through the back-end system.

About this task

Table 8-7. Attributes

Attribute	Mandatory or Optional	Description
`content`	Mandatory	Defines the properties of the entity. Supports JSON and Atom/XML formatting.

Procedure

- To create an entity in JSON format, write the input parameters as shown in the following example.

```
"City": "Midland",
  "Country": "USA",
  "LanguageCode": "3",
  "LocalCurrencyCode": "324",
  "MimeType": "",
  "Name": "Destination Paradise",
  "POBox": "322",
  "PostalCode": "48642",
  "Region": "B",
  "Street": "100 Electric Ave",
  "TelephoneNumber": "5558675309",
  "TravelAgencyID": "0000099",
  "URL": "www.foo.com"
```

- To create an entity in XML format, write the input parameters as shown in the following example.

```
"<?xml version=\"1.0\" encoding=\"utf-8\"?>
  <entry xml:base=\"http:\/\/sapwl01.austin.ibm.com:8003\/sap\/opu\/odata\/iwbep\/RMTSAMPLEFLIGHT_2\/\"
xmlns=\"http:\/\/www.w3.org\/2005\/Atom\" xmlns:m=\"http:\/\/schemas.microsoft.com\/ado\/2007\/08\
/dataservices\/metadata\"
xmlns:d=\"http:\/\/schemas.microsoft.com\/ado\/2007\/08\/dataservices\">

  <id>http:\/\/sapwl01.austin.ibm.com:8003\/sap\/opu\/odata\/iwbep\/RMTSAMPLEFLIGHT_2\
/TravelAgencies('00000099')<\/id>
    <title type=\"text\">TravelAgencies('00000099')<\/title>
    <updated>2014-07-18T14:10:27Z<\/updated>
    <category term=\"RMTSAMPLEFLIGHT_2.TravelAgency\"
scheme=\"http:\/\/schemas.microsoft.com\/ado\/2007\/08\/dataservices\/scheme\"\
/><link href=\"TravelAgencies('00000099')\" rel=\"edit\"
title=\"TravelAgency\"\/>
```

```
   <content type=\"application\/xml\">
     <m:properties>
       <d:TravelAgencyID>00000099<\/d:TravelAgencyID>
<d:Name>Destination Paradise<\/d:Name>
<d:Street>100 Electric Ave<\/d:Street>
<d:POBox>322<\/d:POBox>
<d:PostalCode>48642<\/d:PostalCode>
<d:City>Midland<\/d:City>
<d:Country>USA<\/d:Country>
<d:Region>B<\/d:Region>
       <d:TelephoneNumber>9896002072<\/d:TelephoneNumber>
       <d:URL>www.foo.com<\/d:URL>
<d:LanguageCode>34<\/d:LanguageCode>
       <d:LocalCurrencyCode>324<\/d:LocalCurrencyCode>
       <d:MimeType>xml<\/d:MimeType>
     <\/m:properties>
   <\/content>
 <\/entry>"
```

Results

If you use the previous examples, you receive the following response from MobileFirst Server.

```
{
  "d": {
    "City": "Midland",
    "Country": "USA",
    "LanguageCode": "3",
    "LocalCurrencyCode": "324",
    "MimeType": "",
    "Name": "Destination Paradise",
    "POBox": "322",
    "PostalCode": "48642",
    "Region": "B",
    "Street": "100 Electric Ave",
    "TelephoneNumber": "5558675309",
    "TravelAgencyID": "00000099",
    "URL": "www.foo.com",
    "__metadata": {
    "id": "http:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwbep\/RMTSAMPLEFLIGHT_2\/TravelAgencies('00000099')",
    "type": "RMTSAMPLEFLIGHT_2.TravelAgency",
    "uri": "http:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwbep\/RMTSAMPLEFLIGHT_2\/TravelAgencies('00000099')"
    }
  },
  "isSuccessful": true,
  "responseHeaders": {
    "content-length": "554",
    "content-type": "application\/json; charset=utf-8",
    "dataserviceversion": "2.0",
    "location": "http:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwbep\/RMTSAMPLEFLIGHT_2\
/TravelAgencies('00000099')",
    "server": "SAP NetWeaver Application Server \/ ABAP 731"
  },
  "statusCode": 201,
  "statusReason": "Created"
}
```

Retrieving an entity:

You can retrieve an entity through a back-end system.

About this task

Table 8-8. Attributes

Attribute	Mandatory or Optional	Description
expand	Optional	Indicates that the response from the back-end represents navigation properties inline, rather than referenced. Formatted as a JSON array.
keys	Mandatory	Identifies which entity is to be retrieved based on its key property or properties. Formatted as a JSON object.
select	Optional	Indicates that a response from the back-end is formatted with a subset of specified properties. Formatted as a JSON array.

Procedure

To retrieve an entity in JSON format, write the input parameters as shown in the following example.

```
{
  "keys":{
    "carrid":"LH"
  },
  "select":["carrid", "carrierFlights"],
  "expand":["carrierFlights"]
}
```

Results

If you use the previous example, you receive the following response from MobileFirst Server.

```
{
  "__metadata": {
    "content_type": "image\/gif",
    "edit_media": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/CarrierCollection('LH')\/$value",
    "media_src": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/CarrierCollection('LH')\/$value",
    "type": "RMTSAMPLEFLIGHT.Carrier",
    "uri": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/CarrierCollection('LH')"
  },
  "carrid": "LH",
  "carrierFlights": {
    "results": [
      {
        "CURRENCY": "EUR",
        "FlightCarrier": {
          "__deferred": {
            "uri": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='LH',connid='0400',fldate=datetime'2013-12-21T00%3A00%3A00')\/FlightCarrier"
          }
        },
        "PAYMENTSUM": "209124.00",
        "PLANETYPE": "A310-300",
        "PRICE": "666.00",
        "SEATSMAX": 280,
        "SEATSMAX_B": 22,
        "SEATSMAX_F": 10,
        "SEATSOCC": 267,
        "SEATSOCC_B": 22,
        "SEATSOCC_F": 9,
        "__metadata": {
           "type": "RMTSAMPLEFLIGHT.Flight",
           "uri": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='LH',connid='0400',fldate=datetime'2013-12-21T00%3A00%3A00')"
        },
        "carrid": "LH",
        "connid": "0400",
        "fldate": "\/Date(1387584000000)\/",
        "flightBookings": {
          "__deferred": {
            "uri": "https:\/\/sap4.sapdevelopcenter.com:444\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='LH',connid='0400',fldate=datetime'2013-12-21T00%3A00%3A00')\/flightBookings"
           }

          ...
```

Updating an entity:

You can update an entity from a back-end system.

About this task

Table 8-9. Attributes

Attribute	Mandatory or Optional	Description
updateContent	Mandatory	This attribute is the new set of properties for the specified entity. Any properties that are not defined in **updateContent** are set to an empty string when the update is complete. Supports JSON and Atom/XML formatting.

Procedure

To update an entity in XML format, write the input parameters as shown in the following example.

```
{
  "keys":{
    "TravelAgencyID"
}
```

Results

If you use the previous example, you receive the following response from MobileFirst Server.

```
"City": "Midland",
"Country": "USA",
"LanguageCode": "3",
"LocalCurrencyCode": "324",
"MimeType": "",
"Name": "Destination Paradise",
"POBox": "322",
"PostalCode": "48642",
"Region": "B",
"Street": "100 Electric Ave",
"TelephoneNumber": "5558675309",
"TravelAgencyID": "00000099",
"URL": "www.foo.com"
```

Deleting an entity:

You can delete an existing entity through the back-end system.

About this task

Table 8-10. Attributes

Attribute	Mandatory or Optional	Description
keys	Mandatory	Identifies which entity must be deleted, based on its key properties. Formatted as a JSON object.

Procedure

To delete an entity in JSON format, write the input parameters as shown in the following example.

```
{
  "keys":{
    "TravelAgencyID":"99"
  }
}
```

If you use the previous example, you receive the following response from MobileFirst Server.

```
{
  "isSuccessful": true,
  "responseHeaders": {
    "content-length": "0",
    "dataserviceversion": "2.0",
    "server": "SAP NetWeaver Application Server \/ ABAP 731"
  },
  "statusCode": 204,
  "statusReason": "No Content"
}
```

Results

A successful deletion results in a `204 No Content` response from the server.

Querying an existing entity:

You can search for existing `Collections` within a back-end system.

About this task

Table 8-11. Attributes

Attribute	Mandatory or Optional	Description
expand	Optional	Indicates that the response from the back-end represents navigation properties inline, rather than referenced. Formatted as a JSON array.
filter	Optional	Uses logic operators to indicate that only the matching criteria are returned in the response. Formatted as a `String`. For examples, see: Filter System Query Option.
custom	Optional	Overrides all other input parameters and directly appends this query to your resource path. You can use this parameter to generate more complex queries. For example: `$expand=Products($filter=Date eq null)`.
select	Optional	Indicates that a response from the Gateway is formatted with a subset of specified properties. Formatted as a JSON array.
skip	Optional	Identifies that the first input number of items of a `Collection` are skipped in the response.
top	Optional	Identifies how many items of a `Collection` are returned in a response. Formatted as a non-negative integer, which is enclosed in quotation marks.

Procedure

- To use the **select** parameter for querying an existing entity in JSON format, write the input parameter as shown in the following example.

```
{
  "select": ["TravelAgencyID"]
}
```

- To use the **filter** parameter for querying an existing entity in JSON format, write the input parameter as shown in the following example.

```
{
  "filter": "TelephoneNumber eq '5558675309'"
}
```

- To use the **expand** parameter for querying an existing entity in JSON format, write the input parameter as shown in the following example.

```
{
  "expand": ["carrierFlights"]
}
```

- To use the **skip** parameter for querying an existing entity in JSON format, write the input parameter as shown in the following example.

```
{
  "skip": "2"
}
```

- To use the **top** parameter for querying an existing entity in JSON format, write the input parameter as shown in the following example.

```
{
  "top": "2"
}
```

Retrieving a property of an entity:

You can retrieve a specific property from an entity through a back-end system.

About this task

Table 8-12. Attributes

Attribute	Mandatory or Optional	Description
keys	Mandatory	Identifies the entity to be retrieved based on its key properties. Formatted as a JSON `Object`.
property	Optional	Defines the `Property` to be retrieved. Formatted as a `String`.

Procedure

- To retrieve a property from an entity in JSON format, write the input parameters as shown in the following example.

```
{
  "keys":{
    "carrid":'IBM Air',
    "connid":'0017',
    "fldate":"2013-12-18T00:00:00"
  },
  "property":"carrierFlights"
}
```

If you use the previous example, you receive the following response from MobileFirst Server.

```
{
  "isSuccessful": true,
  "results": [
    {
      "CURRENCY": "USD",
      "FlightCarrier": {
        "__deferred": {
          "uri": "https:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='IBM Air',connid='0017',fldate=datetime'2013-12-18T00%3A00%3A00')\/FlightCarrier"
          }
        },
        "PAYMENTSUM": "192281.41",
        "PLANETYPE": "747-400",
        "PRICE": "422.94",
        "SEATSMAX": 385,
        "SEATSMAX_B": 31,
        "SEATSMAX_F": 21,
        "SEATSOCC": 374,
        "SEATSOCC_B": 28,
        "SEATSOCC_F": 21,
        "__metadata": {
          "type": "RMTSAMPLEFLIGHT.Flight",
          "uri": "https:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='IBM Air',connid='0017',fldate=datetime'2013-12-18T00%3A00%3A00')"
        },
        "carrid": "IBM Air",
        "connid": "0017",
        "fldate": "\/Date(1387324800000)\/",
        "flightBookings": {
          "__deferred": {
            "uri": "https:\/\/servl01.tampa.ibm.com:1234\/sap\/opu\/odata\/iwfnd\/RMTSAMPLEFLIGHT\
/FlightCollection(carrid='IBM Air',connid='0017',fldate=datetime'2013-12-18T00%3A00%3A00')\/flightBookings"
          }
        },
        "flightDetails": {
          "__metadata": {
            "type": "RMTSAMPLEFLIGHT.FlightDetails"
          },
          "airportFrom": "JFK",
          "airportTo": "SFO",

             ...
```

- To retrieve a property from an entity in JSON format, write the input parameters as shown in the following example.

  ```
  {
    "keys":{
      "carrid":'AA',
      "connid":'0017',
      "fldate":"2013-12-18T00:00:00"
    },
    "property":"CARRNAME/$value"
  }
  ```

  If you use the previous example, you receive the following response from MobileFirst Server.

  ```
  {
    "RETURN": "American Airlines",
    "isSuccessful": true,
    "responseHeaders": {
      "content-length": "17",
      "content-type": "text\/plain; charset=utf-8",
      "dataserviceversion": "2.0",
      "server": "SAP NetWeaver Application Server \/ Tampa 1234",
      "x-csrf-token": "9PfsXHsbriIR4PNwfLKBAg=="
    },
    "statusCode": 200,
    "statusReason": "OK"
  }
  ```

Developing JavaScript adapter code:

Learn about implementing a procedure in the adapter XML file, calling Java code from a JavaScript adapter, and invoking a back-end service.

Implementing adapter procedures:

Implement a procedure in the adapter XML file, using an appropriate signature and any return value.

Ensure that you have declared a procedure in the adapter XML file, using a `<procedure>` tag. 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 MobileFirst 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.

Example

This example demonstrates how to use JavaScript on the server side. Note the following when performing procedures on the server side:

- Procedures are implemented in the adapter JavaScript file.
- The service URL is used for procedure invocations.
- Some parts of the URL are constant; for example, http://example.com/. They are declared in the XML file. Other parts of the URL can be parameterized; that is, substituted at run time by parameter values that are provided to the MobileFirst procedure. The following URL parts can be parameterized:
  - Path elements
  - Query string parameters
  - Fragments

For advanced options for adapters, such as cookies, headers, and encoding, see “HTTP adapter `connectionPolicy` element” on page 8-43.

In the JavaScript file, use the same procedure name as in the XML file. The mandatory parameters to call the procedure are method, path, and returnedContentType. The procedure can be parameterized at run time, for example:

```
function getFeeds() {
  var input = {
      method : 'get',
      returnedContentType : 'xml',
      path : "rss.xml"
  };


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

To call an HTTP request, use the WL.Server.invokeHttp method. Provide an input parameter object, which must specify the following options:

- The HTTP method: GET, POST, PUT, or DELETE

- The returned content type: XML, JSON, HTML, or plain
- The service path
- The query parameters (optional)
- The request body (optional)
- The transformation type (optional)

For a complete list of options, see WL.Server class.

Implementing JavaScript adapters to support session-independent mode

In versions earlier than V7.1.0, developers were able to store the applicative state in the HTTP session, either directly by using the session object, namely WL.Server.getClientRequest().getSession() (see WL.Server) or by using a global JavaScript variable.

If you are working in session-independent mode that became available starting with IBM MobileFirst Platform Foundation for iOS V7.1.0, you may not use HTTP sessions to save the applicative state of the adapter. It must be persisted outside the session, for example, by using a database such as Cloudant. For more information on session-independent mode, including steps for upgrading projects that were created in earlier versions of IBM MobileFirst Platform Foundation for iOS, see "Session-independent mode" on page 8-108.

Example: Upgrading application code from pre-V7.1.0 to V7.1.0

The following code snippets provide an example for upgrading an app, ShoppingCartAdapter, from an earlier version of IBM MobileFirst Platform Foundation for iOS so that it support the changes for session-independent mode in V7.1.0

Consider a shopping-cart app in which users can add items to a cart, see the list of items that are already in the cart, and submit an order when satisfied with the items in it.Before V7.1.0, this functionality might have been implemented by the following code:

```
//ShoppingCartAdapter-impl.js

var itemsArray = new Array();

function addItemToCart(item) {
    itemsArray.push(item);

}

function getItemsFromCart() {
    return itemsArray;
}
```

If you adapt the code to work in session-independent mode under V7.1.0 and persist to an external database, such as Cloudant, the code might look as follows:

```
//ShoppingCartAdapter.xml

//Insert cloudant credentials in the domain tag of the adapter.xml
<domain>username.cloudant.com</domain>

//ShoppingCartAdapter-impl.js

function addItemToCart(item) {
    var db = 'shoppingcartdb';

    // Get the clientID from the MFP API (new in 7.1.0)
    var userId = WL.Server.getClientId();

    var response = readFromCloudant(db, userId);

    // See if we found the document
    if(response.cloudantSuccess){
        var doc = response.document;

        // Update the document
        doc.items.push(item);

        // Save the updated document
        return saveOnCloudant(db, doc);
    }else{
        // New Document - save to DB
        var items = new Array();
        items.push(item);
        return saveOnCloudant(db, {_id:userId, 'items':items});
    }
}

function getItemsFromCart() {
    // Get the clientID from the MFP API (new in 7.1.0)
    var userId = WL.Server.getClientId();
    // This will return all documents by this user ID in the shoppingCartsDB
    var response = readFromCloudant('shoppingcartdb', userId);

    return response;
}

/**
* Saves this document on the cloudant DB
* @param db - the cloudant DB to store the document to
* @param document - the document to insert
* @returns
*/
function saveOnCloudant(db, document){
       var input = {
                method : 'post',
                returnedContentType : 'plain',
                path : db,
                headers: {
                    "Authorization":"<your authorization information>"
                },
                body:{
                    contentType:'application/json; charset=UTF-8',
                    content:JSON.stringify(document)
            }};

     return  WL.Server.invokeHttp(input);
}

/**
* Gets the document (If exists) that corresponds to this _id
* @param db - the cloudant DB to get the document from
* @param key - the _id to search by
* @returns
*/
function readFromCloudant(db, key){
     var input = {
                method : 'get',
                returnedContentType : 'plain',
                path : db + "/" + key,
                headers: {
                    "Authorization":"<your authorization information>"
                }
            };
     var response = WL.Server.invokeHttp(input);

     // Document was found - Note this is not enough to only check for 200
     if(response.statusCode == 200){
         return {cloudantSuccess:true, document:JSON.parse(response.text)};
     }
     else{
         return {cloudantSuccess:false};
     }

}
```

Adapting custom security code to support session-independent mode

In versions earlier than V7.1.0, developers were able to store custom security state in the HTTP session.

In session-independent mode that became available starting with IBM MobileFirst Platform Foundation for iOS V7.1.0, you may not use the HTTP session to save state. Custom security context must therefore be stored elsewhere.

For more information on session-independent mode, see "Session-independent mode" on page 8-108.

This section describes the three main forms of customized security features: adapter-based authentication, custom Java authenticator or login module, and event source (push and geolocation) flows.

Adapter-based authentication

This example shows how to create a double-step authentication flow with adapter-based authentication to support session-independent mode. The example uses a third-party database, in this case, Cloudant and the MobileFirst WL.Server.getClientId() API.

See Figure 1: the adapter requests the ID that is associated with the currrent client from MobileFirst Serverand uses it when reading or writing the custom security context to or from Cloudant.

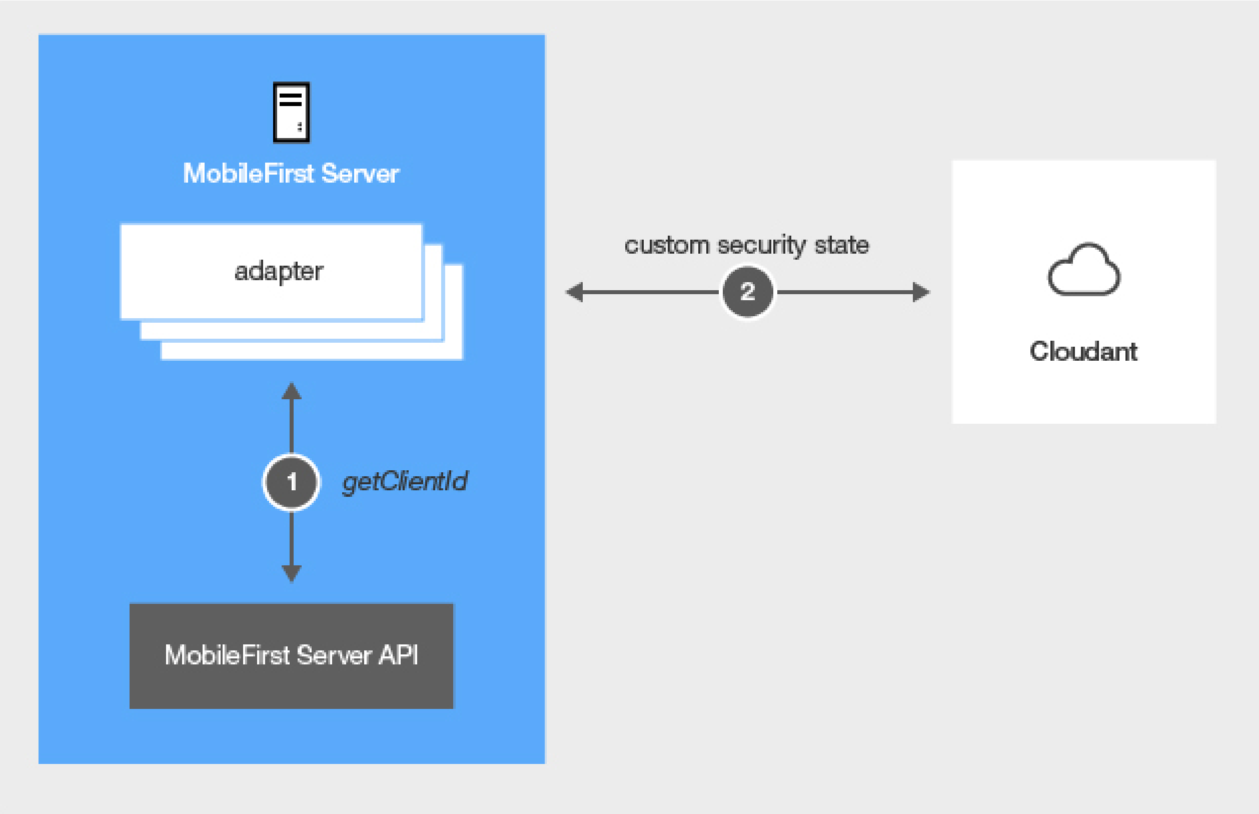


Figure 8-17. Double-step authentication flow with `WL.Server.getClientId()`

Here's the code:

```
var dbName = "REPLACE_ME_WITH_THE_DATABASE_NAME";
var auth = "Basic REPLASE_ME_WITH_THE_BASE-64_ENCODED_STRING";

function onAuthRequired(headers, errorMessage){
    errorMessage = errorMessage ? errorMessage : null;
    return {
        authRequired: true,
        authStep: 1,
        errorMessage: errorMessage
    };
}

function submitAuthenticationStep1(username, password){
    if (username === "user" && password === "password"){
        WL.Logger.debug("Step 1 :: SUCCESS");
        // Get the internal MFP Unique ClientID (New in 7.1.0)
var clientId = WL.Server.getClientId();
        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {}
        };

        //Validate that the DB doesn't already contains the ClientId
        var response = deleteUserIdentityFromDB(dbName, null);

        //Write ClientId to DB
        var response = writeUserIdentityToDB(dbName, {_id:clientId, "userIdentity":u
        if (response){
            return {
                authRequired: true,
                authStep: 2,
                question: "What is your pet's name?",
                errorMessage : ""
            };
        } else {
            return onAuthRequired(null, "Database ERROR");
        }

    } else{
        WL.Logger.debug("Step 1 :: FAILURE");
        return onAuthRequired(null, "Invalid login credentials");
    }
}

function submitAuthenticationStep2(answer){
    // Get the internal MFP Unique ClientID (New in 7.1.0)
     var clientId = WL.Server.getClientId();
    var response = readUserIdentityFromDB(dbName, clientId);
    if (response){
        if (answer === "Lassie"){
            var doc = JSON.parse(response.text);
            var userIdentity = doc.userIdentity;
            WL.Logger.debug("Step 2 :: SUCCESS");
            WL.Server.setActiveUser("DoubleStepAuthRealm", userIdentity);
            WL.Logger.debug("Authorized access granted");

            var response = deleteUserIdentityFromDB(dbName, doc);

            return {
                authRequired: false
            };
        } else{
            WL.Logger.debug("Step 2 :: FAILURE");
            return onAuthRequired(null, "Wrong security question answer");
        }
    } else {
        WL.Logger.debug("Step 1 :: FAILURE");
        return onAuthRequired(null, "Database ERROR");
    }

}

function getSecretData(){
    return {
        secretData: "Very very very very secret data"
    };
}

function onLogout(){
    WL.Logger.debug("Logged out");
}


function writeUserIdentityToDB(db, document){
       var input = {
                   method : 'post',
                returnedContentType : 'plain',
                path : db,
                headers: {
                    "Authorization":auth
                },
                body:{
                    contentType:'application/json; charset=UTF-8',
                    content:JSON.stringify(document)
            }};

         var response = WL.Server.invokeHttp(input);
         var responseString = "" + response.statusCode;

        //Checking if the invocation was successful - status code = 2xx
         if (responseString.indexOf('2') === 0){
             return response;
         }
         return null;
}


function deleteUserIdentityFromDB(db, document){
    var doc = document;
    if (!doc){
        var clientId = WL.Server.getClientId();
        var response = readUserIdentityFromDB(dbName, clientId);
        if(!response){
            return;
        } else {
            doc = JSON.parse(response.text);
        }
    }
    var id = doc._id; // The id of the doc to remove
    var rev = doc._rev; // The rev of the doc to remove
    var input = {
                method : 'delete',
                returnedContentType : 'plain',
                path : db + "/" + id + "?rev=" + rev,
                headers: {
                    "Authorization":auth
                    }
            };
     return WL.Server.invokeHttp(input);

}

function readUserIdentityFromDB(db, key){
     var input = {
                method : 'get',
                returnedContentType : 'plain',
                path : db + "/" + key,
                headers: {
                    "Authorization":auth                }
            };
     var response = WL.Server.invokeHttp(input);
     var responseString = "" + response.statusCode;

     //Checking if the invocation was successful - status code = 2xx
     if (responseString.indexOf('2') === 0){
         return response;
     }
     return null;
}
```

Custom Java authenticator/login module

For more information, see “Implementing Java-based custom authenticators” on page 8-278.

Event sources

Event sources are adapters that are called indirectly, as a result of a push event, or a geolocation trigger. Keep in mind that event sources in a session-independent configuration cannot store information in an HTTP session, as described in “Session-independent mode” on page 8-108.

isSuccessful is a reserved JSON object key

isSuccessful is a reserved key name in the JSON object that is used by the

MobileFirst server-client protocol. You cannot use isSuccessful as a key name in the JSON object that is returned in any of your adapter procedures.

The Rhino container:

IBM MobileFirst Platform Foundation for iOS 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.

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 MobileFirst project under *`Worklight Project Folder`*`/server/java`. They are automatically built and deployed to the MobileFirst Server, and the result of the build is placed under *`Worklight Project Folder`*`/bin`

Example

```
var x = new MyJavaClass();
var y = x.myMethod(1, "a");
```

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.

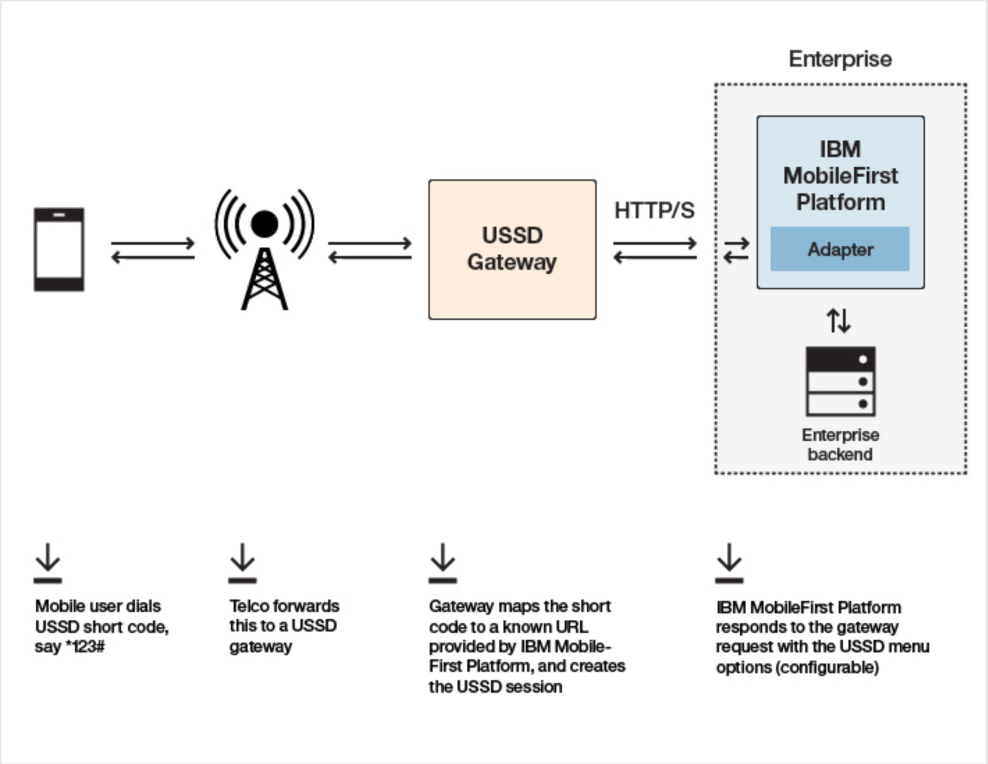
USSD establishes a real-time session between the mobile phone and the application that handles the service.

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 IBM MobileFirst Platform Foundation for iOS, creates the USSD session, and forwards the request to the URL.
4. A MobileFirst adapter with the matching filter receives the request and responds to the gateway request with the configurable USSD menu/simple text.


Enterprise
IBM MobileFirst Platform
Adapter
USSD Gateway
HTTP/S
Enterprise backend
Mobile user dials USSD short code, say *123#
Telco forwards this to a USSD gateway
Gateway maps the short code to a known URL provided by IBM Mobile-First Platform, and creates the USSD session
IBM MobileFirst Platform responds to the gateway request with the USSD menu options (configurable)

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 MobileFirst 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 MobileFirst 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, MobileFirst 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 MobileFirst HTTP adapter to test the USSD functionality.

Before you begin

This feature is only available within MobileFirst 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 MobileFirst 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 MobileFirst Back-end Service**.

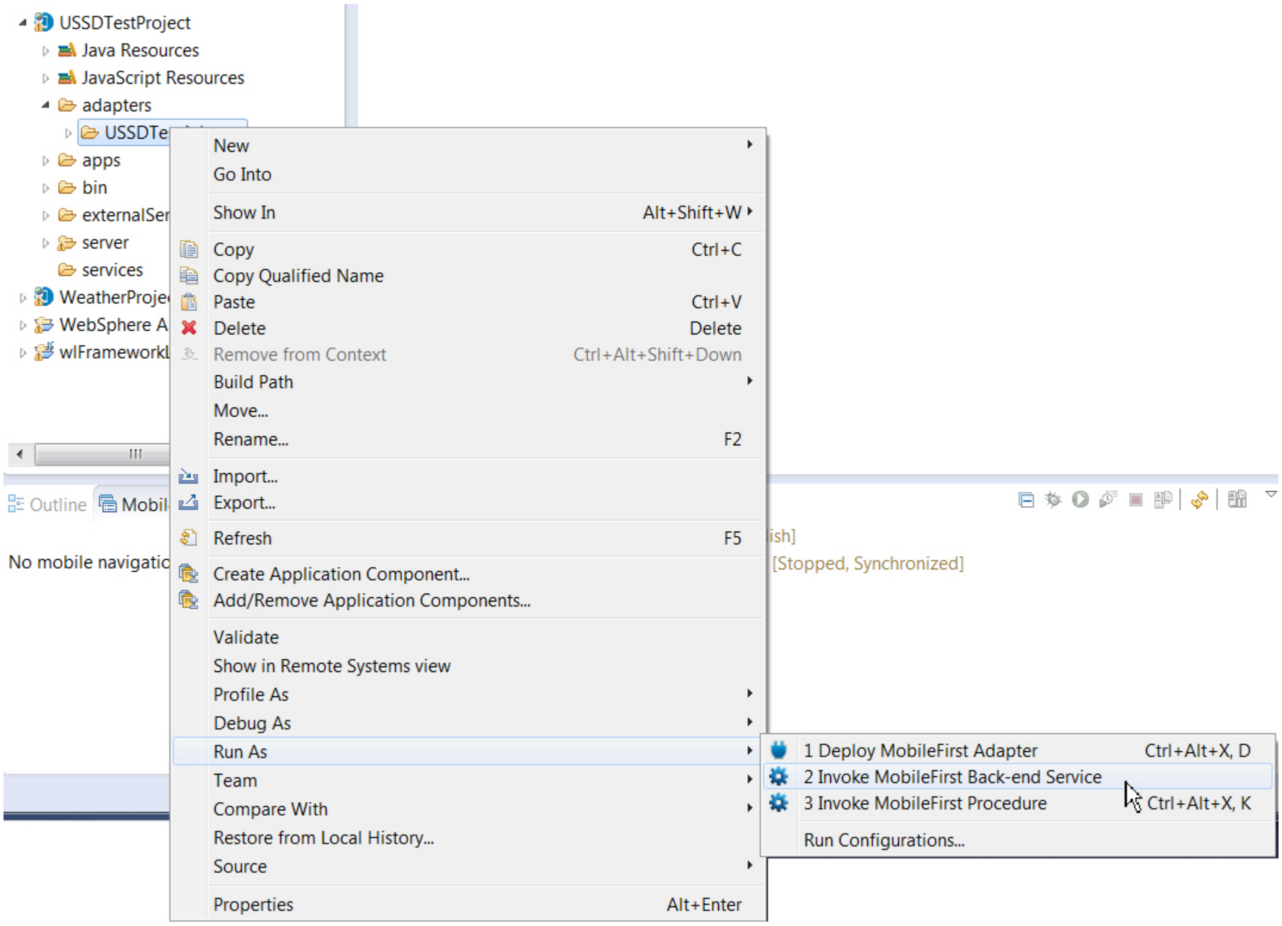
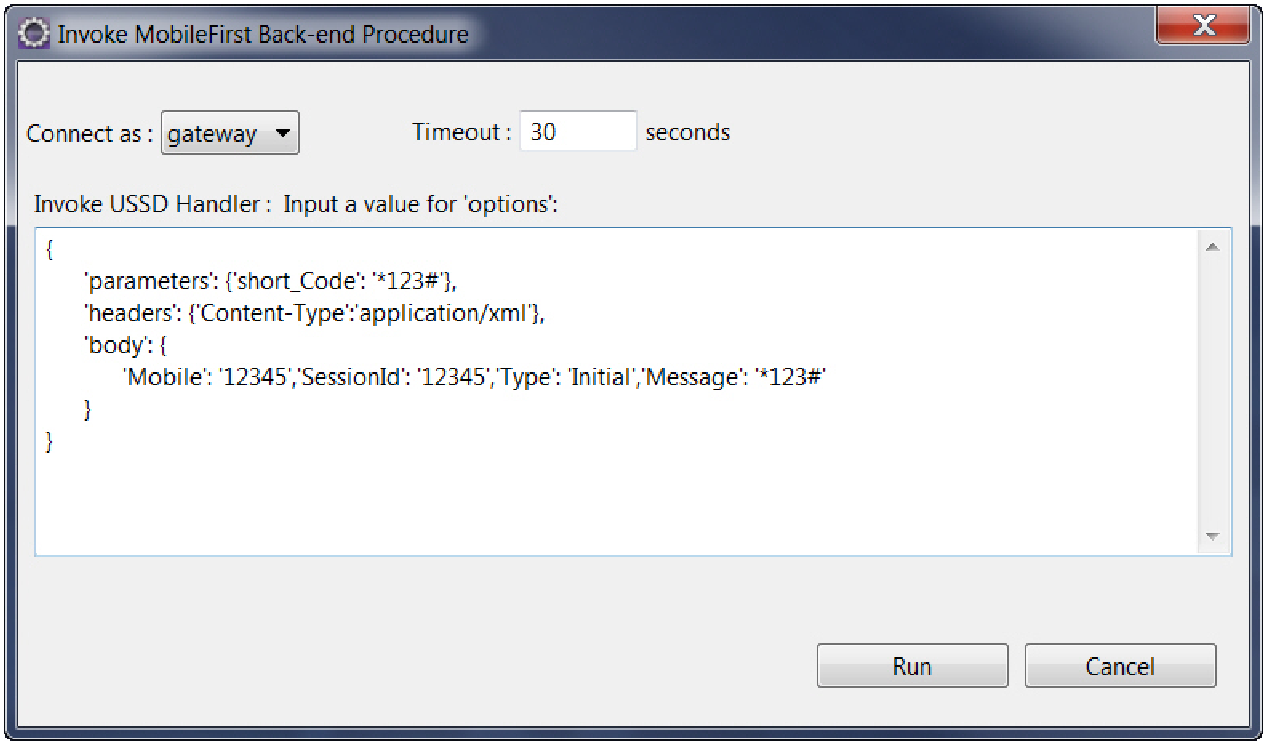


Figure 8-18. Invoking a MobileFirst back-end service

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.



Connect as : gateway   Timeout : 30 seconds

Invoke USSD Handler : Input a value for 'options':

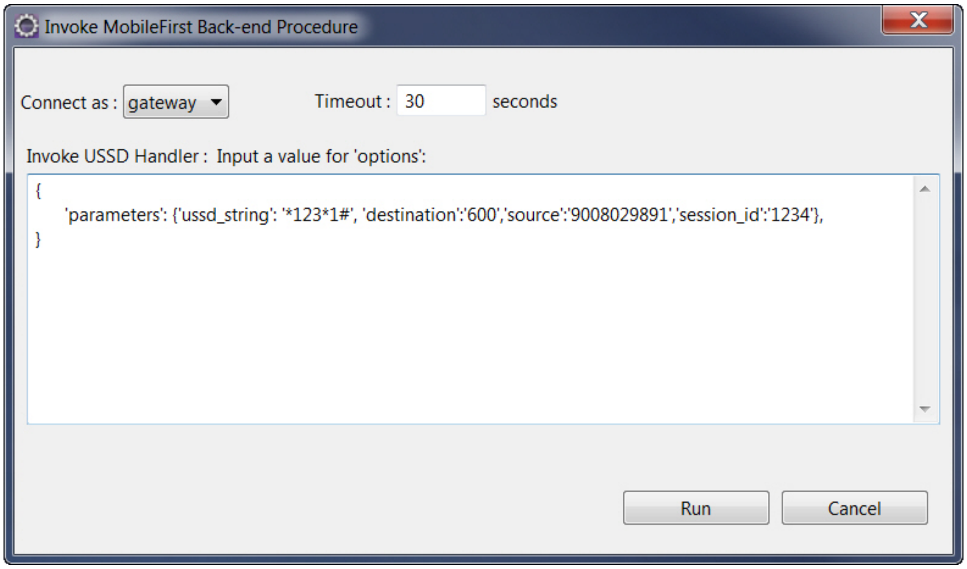
```
{
    'parameters': {'short_Code': '*123#'},
    'headers': {'Content-Type':'application/xml'},
    'body': {
        'Mobile': '12345','SessionId': '12345','Type': 'Initial','Message': '*123#'
    }
}
```

Figure 8-19. 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.

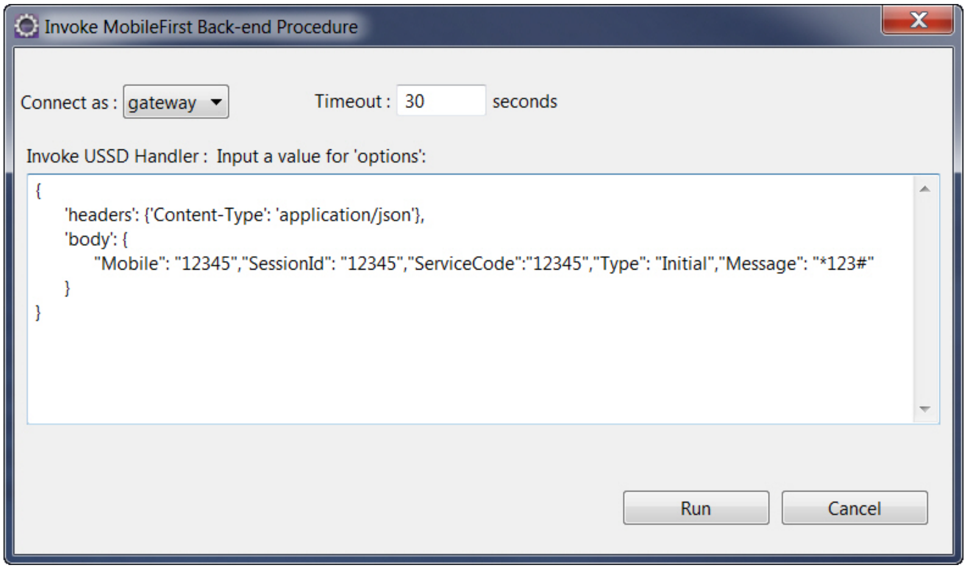


Connect as : gateway   Timeout : 30 seconds

Invoke USSD Handler : Input a value for 'options':

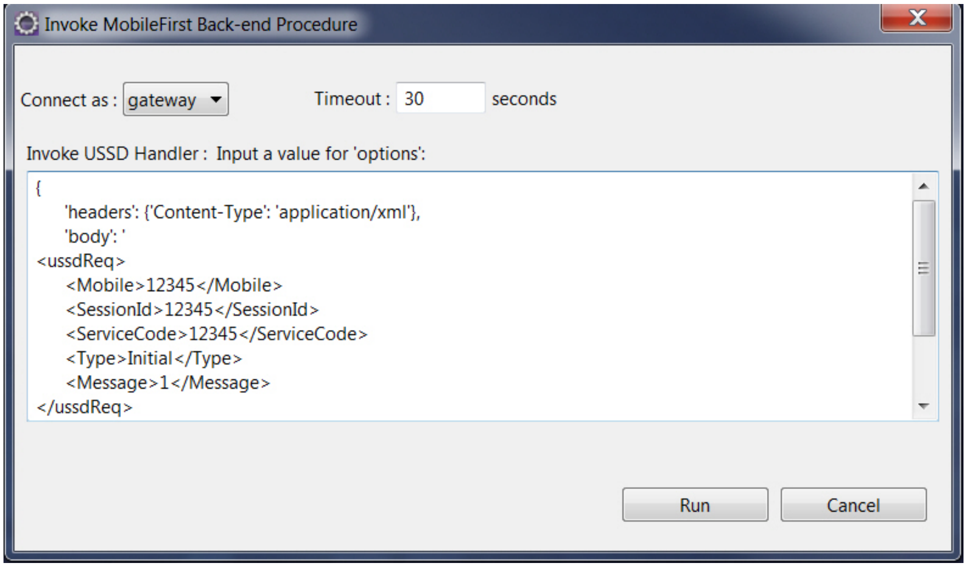
```
{
    'parameters': {'ussd_string': '*123*1#', 'destination':'600','source':'9008029891','session_id':'1234'},
}
```

Example 2: Passing JSON parameters in the body.


Invoke MobileFirst Back-end Procedure
Connect as : gateway
Timeout : 30 seconds
Invoke USSD Handler : Input a value for 'options':
Run
Cancel

```
{
    'headers': {'Content-Type': 'application/json'},
    'body': {
        "Mobile": "12345","SessionId": "12345","ServiceCode":"12345","Type": "Initial","Message": "*123#"
    }
}
```

Example 3: Passing XML in the body.


Invoke MobileFirst Back-end Procedure
Connect as : gateway
Timeout : 30 seconds
Invoke USSD Handler : Input a value for 'options':
Run
Cancel

```
{
    'headers': {'Content-Type': 'application/xml'},
    'body': '
<ussdReq>
    <Mobile>12345</Mobile>
    <SessionId>12345</SessionId>
    <ServiceCode>12345</ServiceCode>
    <Type>Initial</Type>
    <Message>1</Message>
</ussdReq>
```

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 ({ }).

Deploying adapters

In MobileFirst Studio, you can automatically deploy a new or modified adapter to the MobileFirst Server.

Procedure

Right-click the adapter folder and select **Run As** > **Deploy MobileFirst Adapter**.

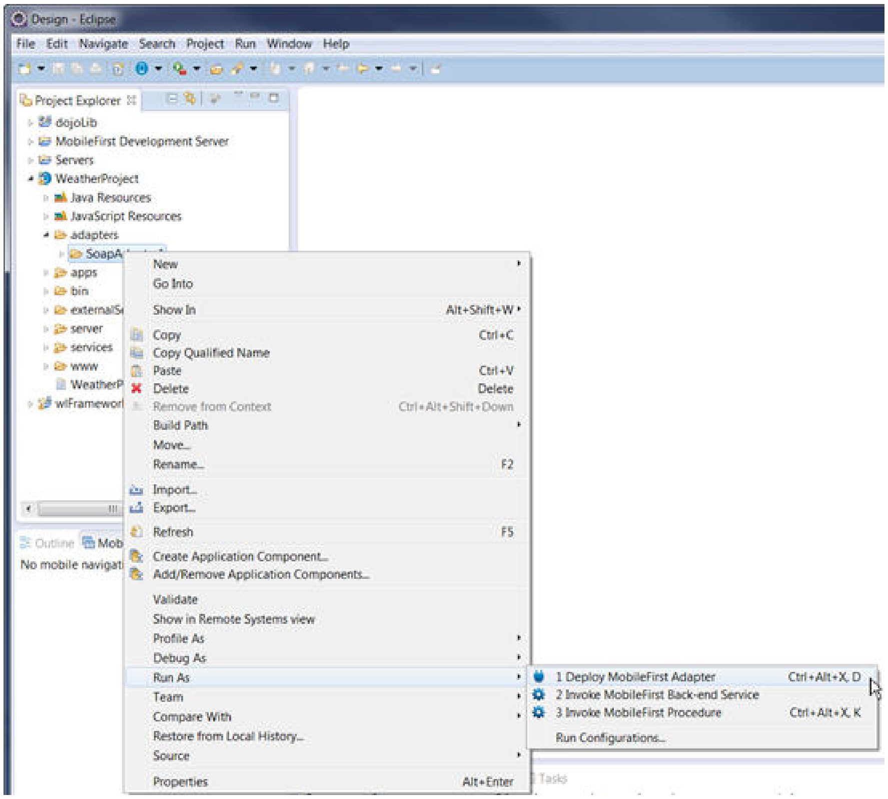


Figure 8-20. Deploying a MobileFirst adapter.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

Note: When the development server is started, adapters are automatically deployed after they are created or modified and saved. You can view this feature by clicking **Window** > **Preferences**. Select **MobileFirst**. To change this feature, clear the **Automatically Deploy Adapters on Change** check box. The default value of this feature is **`true`**. The **`preview`** command automatically deploys all adapters that are not deployed for a project if this preference is set to **`true`**.

Testing adapters

You can select a procedure, enter a set of parameters, and invoke the procedure on the MobileFirst Server.

Testing adapters with MobileFirst Platform Command Line Interface for iOS (CLI)

To test an adapter by using CLI, you must call the adapter.

To call an adapter by using CLI, run the **`mfp adapter call`** command. For more information about this command, see mfp adapter call.

Testing adapters with external tools such as Postman

MobileFirst Java adapters expose a full REST API. If you know the URL of a resource or procedure, you can use HTTP tools such as Postman to test requests and pass the following parameters.

- URL parameters

- Path parameters
- Body parameters
- Headers

If your resource is protected by a security scope, the request prompts you to provide a valid authorization header. The default value of this security scope is simple. Unless you disabled security, the endpoint is protected.

As a solution for testing, the development version of the Worklight Server includes a test token endpoint.

To receive a Test Token, create an HTTP `POST` request to `http(s)://{server_ip}:{server_port}/{project_name}/authorization/v1/testtoken` with your HTTP client (Postman).

You receive a JSON object with a temporary valid authorization token.

```
{
  "Authorization": "Bearer eyJhbGciOiJSUzI1NiIsImpwayI6eyJhbGciOiJSU0EiLCJleHAiOiJBUUFCIiwib
QU0wRGQ3eEFkdjZILXlnTDdyOHFDTGRFLTNJMm2FPZUlxb2UtckpBMHVadXcyckhoWFozV1ZDZUtlelJWY0NPWXNRTi1
UUswbWZ6NV8zby1ldjBVWXdYa1NPd0JCbDFFaHFJd1ZEd09pZWcySk1HbDBFWHNQWmZrTlpJLUhVNG9NaWktVHJOTHp.
}
```

In your next requests to the adapter endpoints, add an HTTP header with the name `Authorization` and the value that you received that starts with `Bearer`.

Now that you have a valid token, you can access the resource by using the security framework.

For more information about the test token endpoint, see "The test token endpoint" on page 8-208.

Session-independent mode

Starting with V7.1.0, MobileFirst applications run by default in session-independent mode.

Overview

In previous versions of the product, the authentication context of clients (for example, whether they are logged in to a realm) was wholly dependent on an HTTP session. After logging in, the end user remained logged in only as long as the HTTP session was alive. Whenever the session ended, the user was automatically logged out. In addition, to ensure the continuity of an HTTP session, "sticky" sessions were required when using a load balancer.

The session-independent mode in V7.1.0 decouples the link between authentication context and HTTP sessions by using the attribute store module for saving the authentication context. The interaction between the client and MobileFirst Server no longer has to be session-dependent. IT can auto-scale the network as needed and use a load-balancer without requiring sticky HTTP sessions.

One advantage of this feature is that starting from V7.1.0, if your system is operating in session-independent mode and you choose to scale back or scale up any number of servers, the user experience is unaffected.

Restriction:

- By default, new projects that are created in V7.1.0 are *session independent*. Old projects, even after upgrade remain session dependent. For steps to upgrade older apps to work in session-independent mode in V7.1.0, see Upgrading projects to work in session-independent mode.

- Session-independent mode is not supported in configurations of IBM MobileFirst Platform Foundation for iOS that include any of the following components or environments:
  - Blackberry
  - Windows Phone Silverlight 8 (native)
  - Adobe AIR
  - Desktop browser
  - Mobile web apps
  - JavaME native
  - Geolocation
  - USSD

Attribute store types for session-independent mode

The attribute store is where the authorization server stores the authentication context of clients (such as the state of authenticators and login modules).

There are three types of attribute store configurations that support session-independent mode: attribute store over the runtime database (default), attribute store over IBM WebSphere eXtreme Scale, and attribute store over IBM Data Cache for Bluemix.

Note: The attribute store type that you select has an impact on the scalability and performance of the IBM MobileFirst Platform Server. In general, when high throughput is required, optimal results can be achieved by using the attribute store over IBM WebSphere eXtreme Scale. To choose the configuration that best fits your throughput and scalability requirements, see the Developer Center website for IBM MobileFirst Platform Foundation.

Attribute store over the runtime database (default)

Starting from V7.1.0, the attribute store in IBM MobileFirst Platform Server is persisted by default to the runtime database (Cloudant or relational database). Figure 1 illustrates a server cluster that employs this configuration.

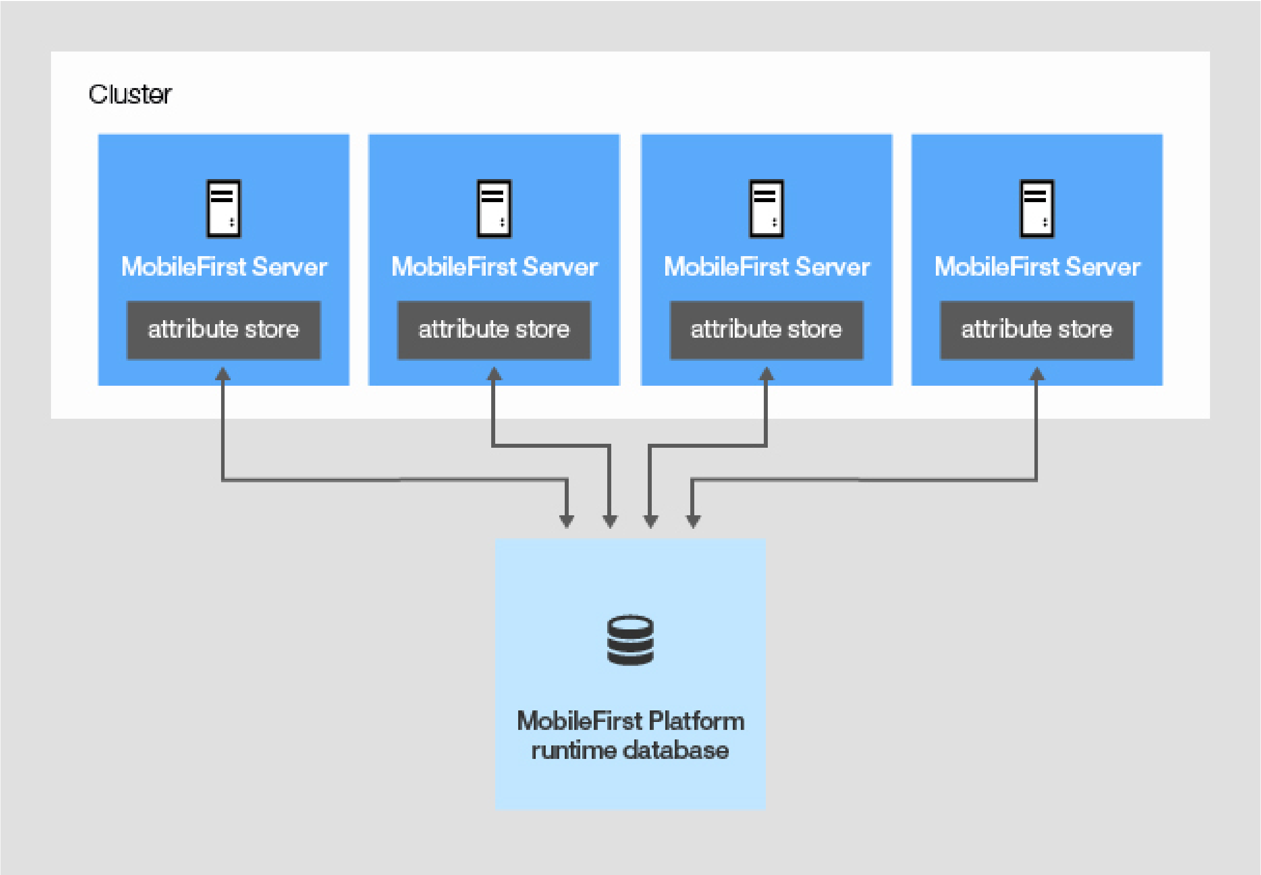


Figure 8-21. Attribute store over runtime database

The following settings in the `worklight.properties` file determine the default behavior:

- `mfp.attrStore.type` is set to `database` (default)
- `mfp.session.independent` is set to `true` (default)

Attribute store over IBM WebSphere eXtreme Scale

The attribute store can be cached in eXtreme Scale, with or without a Loader plug-in for persistence to an SQL or Cloudant database. For more information, see "Attribute store over IBM WebSphere eXtreme Scale" on page 8-112.

Attribute store over IBM Data Cache for Bluemix

The attribute store data for applications that are deployed in IBM Containers can be cached to IBM Data Cache. For more information, see "Deploying to the cloud in an IBM Container" on page 11-107 and IBM Data Cache for Bluemix.

Important:

- In a session-independent configuration, a request can be processed by one server in a cluster and the next request by another. This means that if you apply a fix pack or other change to a farm, particularly if the fix pack includes changes to authentication, all servers must be upgraded at the same time. If the fix pack is not applied to the entire farm at once, requests may fail as they move back and forth between servers where the fix pack has been applied and servers where it has not.

- The attribute store module uses serialized objects, so the Java Virtual Machine (JVM) of any server must be able to read what the JVM of other servers has serialized. Ensure that all members of a cluster use compatible serialization.

Session-dependent mode

V7.1.0 still allows you to run applications in session-dependent mode, where the attribute store is held in the HTTP session. To work in session-dependent mode, set the `mfp.attrStore.type` property to `httpsession`. Also, ensure that you have set the `mfp.session.independent` property to `false`, or MobileFirst Server will not come up and an error will be reported. See “Configuring session dependency” on page 11-54.

Upgrading projects to work in session-independent mode

Projects that are upgraded to work with MobileFirst Server V7.1.0 from earlier versions will remain in *session-dependent* mode by default.

For more information about upgrading old projects so that they work in session-independent mode, see Upgrading projects to work in session-independent mode.

Developing apps to work in IBM MobileFirst Platform Foundation for iOS V7.1.0

- In previous versions, certain components of your projects could persist data in the HTTP session. This is no longer possible in session-independent mode:
  - **IBM MobileFirst Platform Foundation for iOS security framework**: In session-independent mode, the attribute store is used to save the security state, instead of the HTTP session, which was used in previous versions.
  - **Applicative state and custom security state**: If your existing applications store applicative state or custom security state in the HTTP session, you must adapt your code so that the applicative or security state is held elsewhere

  For guidelines on refactoring existing apps away from HTTP sessions see the following topics:
  - Implementing JavaScript adapters to support session-independent mode
  - Implementing security-based code to support session-independent mode
  - Saving applicative state between requests in Java RESTful adapters
- When implementing or upgrading a custom authenticator, refer to the custom-authenticator guidelines for session-independent mode.

Realm expiration

It is important to note that realm expiration is not new in V7.1.0. In previous versions, it was limited by the HTTP session lifespan, and starting from V7.1.0, it is not. To enforce a lifetime for a client's logged-in state, you must configure the expiration of security realms. When the realm expires, the login session ends and the client app is forced to log in again if it tries to access a resource that is protected by this realm, or if it requests a token that contains this realm.

Version 7.0.0 introduced the optional `expirationInSeconds` configuration element, which you could use for each login module to define the expiration of its realm. Its default setting was -1 to indicate that there was no expiration period and that the login would remain valid until the end of the HTTP session.

Starting with V7.1.0, the login period is entirely dependent on the realm expiration, so **the default value for `expirationInSeconds` is now 3600 seconds** (one hour) and in session-independent mode, it is no longer possible to set the value to -1. For more information, see "Configuring login modules" on page 8-263.

Attribute store over IBM WebSphere eXtreme Scale

To support session-independent mode, you can cache authentication context by using eXtreme Scale.

About this task

Instructions are provided here for configuring an IBM MobileFirst Platform Server cluster to work with eXtreme Scale (Figure 1). It also provides instructions for persisting the cached data to relational and non-relational stores by using the Loader plug-ins that are provided with the product (see Figure 2 and Figure 3).

For more information about session-independent mode, see "Session-independent mode" on page 8-108.

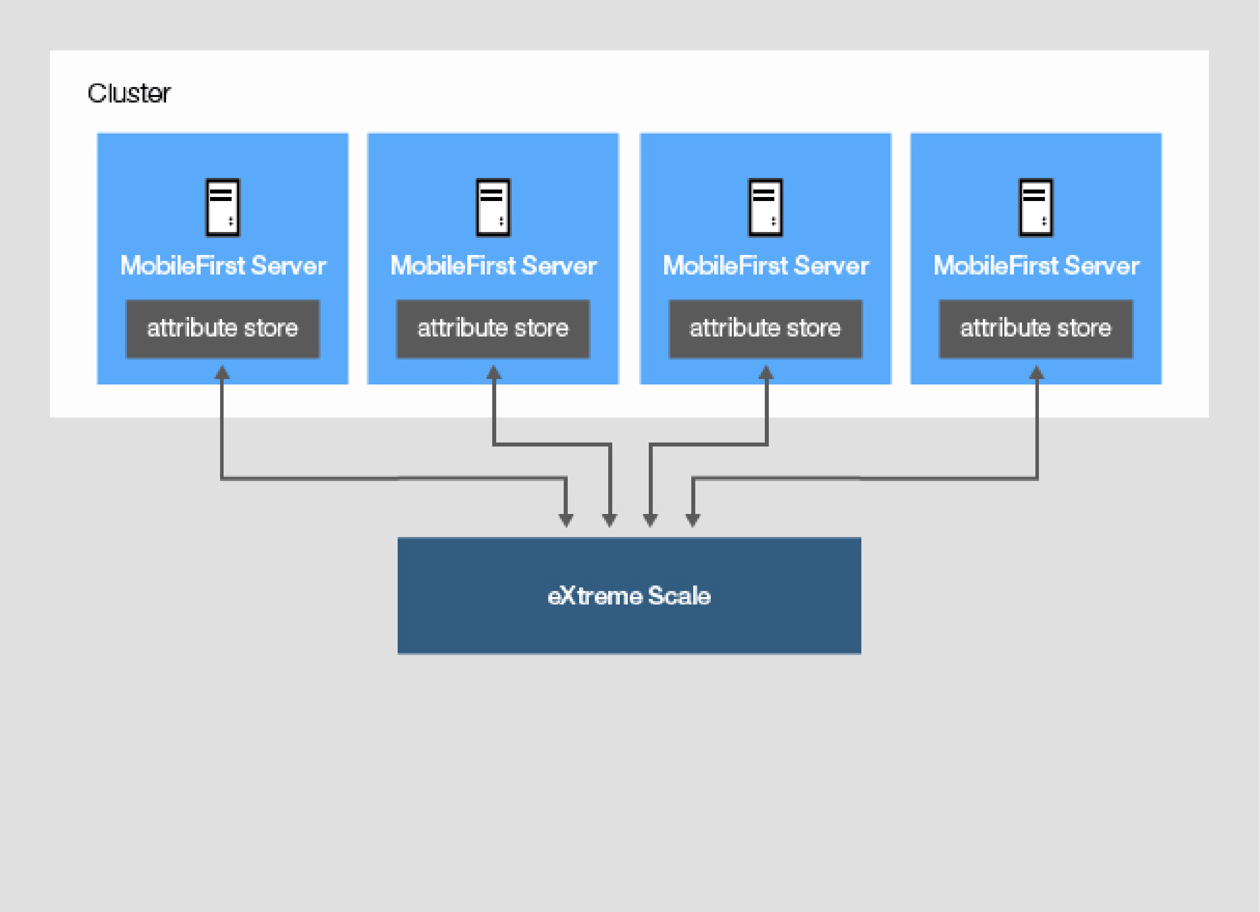


Figure 8-22. Attribute store over eXtreme Scale

eXtreme Scale Grid Definition

About this task

In the eXtreme Scale server, define a dedicated object grid and a backing map for usage of the attribute store table in IBM MobileFirst Platform Server. The recommended eXtreme Scale configurations are described in detail in the WebSphere eXtreme Scale V8.6 Knowledge Center.

Procedure

In the `objectgrid.xml` file, define a dedicated object grid and backing map. Here is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="MFPGrid" txTimeout="30">
    <backingMap name="AttrStoreMap" copyMode="COPY_TO_BYTES"
        lockStrategy="PESSIMISTIC"
        nullValuesSupported="true" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="10" nearCacheEnab
    </objectGrid>
  </objectGrids>
</objectGridConfig>
```

Optionally, the map can be configured to use a Loader plug-in that persists the data. For more information, see “Persisting the eXtreme Scale cache to the IBM MobileFirst Platform Server relational database” on page 8-114.

IBM MobileFirst Platform Server configuration

About this task

In configuring IBM MobileFirst Platform Server, you must define several configuration properties that contain the eXtreme Scale connection parameters and add the eXtreme Scale client library JAR file to the class path.

Procedure

1. Define the following properties in the `worklight.properties` file or by using JNDI:

   ```
   //Enable session independent mode
   mfp.session.independent=true

   //Indicates attribute store must be cached in eXtreme Scale
   mfp.attrStore.type=eXtremeScale

   //A comma-separated list of host-port pairs of the WebSphere eXtreme Scale catalog servers, in
   mfp.attrStore.xs.endpoint=9.111.27.160:2809

   //The user name and password for connecting to the eXtreme Scale server.  Leave empty if authe
   mfp.attrStore.xs.username=
   mfp.attrStore.xs.password=

   //The names of the grid and backing map for the attribute store, as defined in objectgrid.xml
   mfp.attrStore.xs.gridname=MFPGrid
   mfp.attrStore.xs.mapname=AttrStoreMap
   ```

2. Add the eXtreme Scale client library JAR file to the class path of your MobileFirst project runtime application. For Liberty Profile, use `ogclient.jar` and for WebSphere Application Server, use `wsogclient.jar`. Both JAR files are

in the eXtreme Scale installation folder, which is at <*eXtreme Scale_install_folder*>\ObjectGrid\lib by default.

- **In Liberty**
  a. Create a folder WXS under the Liberty shared resources folder and copy the JAR file to that folder.
  b. Add the location of the JAR file to the class loader configuration of your MobileFirst project.

```
<!-- Declare the IBM Worklight project runtime application. -->
<application id="worklight" name="worklight" location="worklight.war" type="war">
    <classloader delegation="parentLast">
        <privateLibrary id="worklightlib_worklight">
            <fileset dir="${shared.resource.dir}/worklight/lib" includes="worklight-jee-lib
            <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_
            <fileset dir="${shared.resource.dir}/WXS" includes="ogclient.jar"/>
        </privateLibrary>
    </classloader>
</application>
```

  Note: When you use eXtreme Scale client with the IBM MobileFirst Platform Server, a known JMX connection issue prevents the use of Oracle JDK. Use IBM JDK, as follows:
  a. In the IBM MobileFirst Platform Server folder, for example: /opt/IBM/WebSphere/Liberty/usr/servers/<*my_username*>, edit the jvm.options file so that it contains the following lines. Replace the yourServerName folder with the one in your setup:

```
-Dcom.ibm.ws.jmx.connector.client.rest.readTimeout=180000
-Djavax.net.ssl.trustStore=/opt/IBM/WebSphere/Liberty/usr/servers/<yourServerName>/resource
-Djavax.net.ssl.trustStorePassword=worklight
-Djavax.net.ssl.trustStoreType=jks
-Djava.security.properties=./java.security.props
```

  b. In the same folder as the server.xml file, create a new file, java.security.props that contains the following two lines. Note that each SSL key has an empty value.

```
ssl.SocketFactory.provider=
ssl.ServerSocketFactory.provider=
```

- **In WebSphere Application Server full profile**:

  Copy the eXtreme Scale JAR file, wsogclient.jar to a local folder and add the path of this JAR file to the class path that is defined in the WL_PLATFORM_LIB shared library.

Persisting the eXtreme Scale cache to the IBM MobileFirst Platform Server relational database

About this task

eXtreme Scale can back up its data to a relational database with a Loader plug-in. For more information about eXtreme Scale and loaders, see Plug-ins for communicating with databases.

In the configuration that is described in this section, the built-in MobileFirst relational database Loader is used to back up eXtreme Scale data to the IBM MobileFirst Platform Server SQL database.

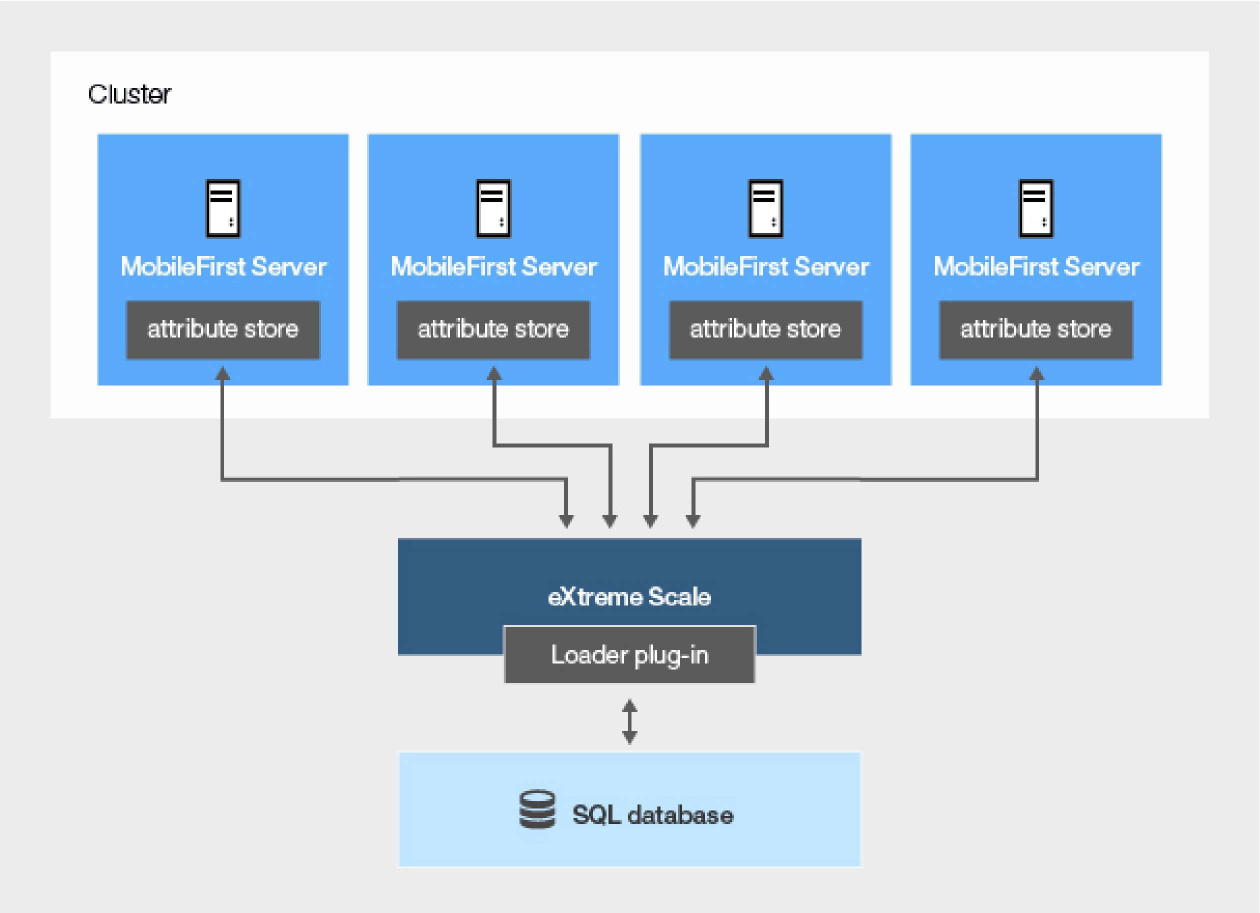


Figure 8-23. Attribute store over eXtreme Scale with SQL Loader plug-in

Note: IBM MobileFirst Platform Server also provides a default write-behind Loader plug-in for backing up eXtreme Scale data with Cloudant. For more information about configuring Cloudant as your database, see “Persisting the eXtreme Scale cache to Cloudant” on page 8-116.

Procedure

1. Find the Loader JAR file, `mfp-xs-loader.jar` in your MobileFirst Server installation folder. By default, it is in `<`*`mfp_install_dir`*`>/WorklightServer/external-server-libraries`.
2. Extract the `meta-inf/persistence.xml` file from the JAR file and replace the `<properties>` tag in the file according to the following example. This example applies to a DB2 database. For other types of database like MySQL or Oracle, you must change the driver class name to the class name for your database type.

```
<properties>
   <property name="openjpa.ConnectionProperties" value="DriverClassName=com.ibm.db2.jcc.DB2Dri
      Url=jdbc:db2://database_host_name:database_port/database_name, MaxActive=100, MaxWait=10
      Username=database_user_ID, Password=databse_user_password"/>
   <property name="openjpa.ConnectionDriverName" value="org.apache.commons.dbcp.BasicDataSourc
</properties>
```

   Replace *database_host_name*, *database_port*, *database_name*, *database_user_ID*, and *databse_user_password* with values for your database.
3. Repackage the JAR file.

Note: As an alternative to providing these parameters in persistence.xml, you can override the parameters when you start the eXtreme Scale Server by using JVM arguments.

4. Add the mfp-xs-loader.jar to the classpath of your eXtreme Scale deployment.
5. Add the database driver JAR file to the classpath of your eXtreme Scale deployment.
6. In objectgrid.xml, add an entry with className="com.worklight.core.persistence.xs.AttributeStoreDBLoader" for backingMapPluginCollection

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="MFPGrid" txTimeout="30">
    <backingMap name="AttrStoreMap" copyMode="COPY_TO_BYTES" pluginCollectionRef="as"
        lockStrategy="PESSIMISTIC" writeBehind="T30;C20"
        nullValuesSupported="true" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="10" nearCacheEna
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="as">
        <bean id="Loader" className="com.worklight.core.persistence.xs.AttributeStoreDBLoader"/>
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

The writeBehind XML attribute is optional.

7. In deployment.xml, increase the **numberOfPartitions** attribute to 50. When you increase the number of partitions, you increase the number of loader threads connecting to the database. An increase in the number of loader threads improves the performance of persisting to the database.

Persisting the eXtreme Scale cache to Cloudant

About this task

In the configuration that is described in this section, the built-in MobileFirst Cloudant database Loader is used to persist eXtreme Scale data to a Cloudant database. This configuration can use either Cloudant Local, or Cloudant DBaaS.

Note: As an alternative, you can use the MobileFirst relational Loader to connect eXtreme Scale with the MobileFirst SQL database. For more information, see Persisting the eXtreme Scale cache to the MobileFirst Server relational database.

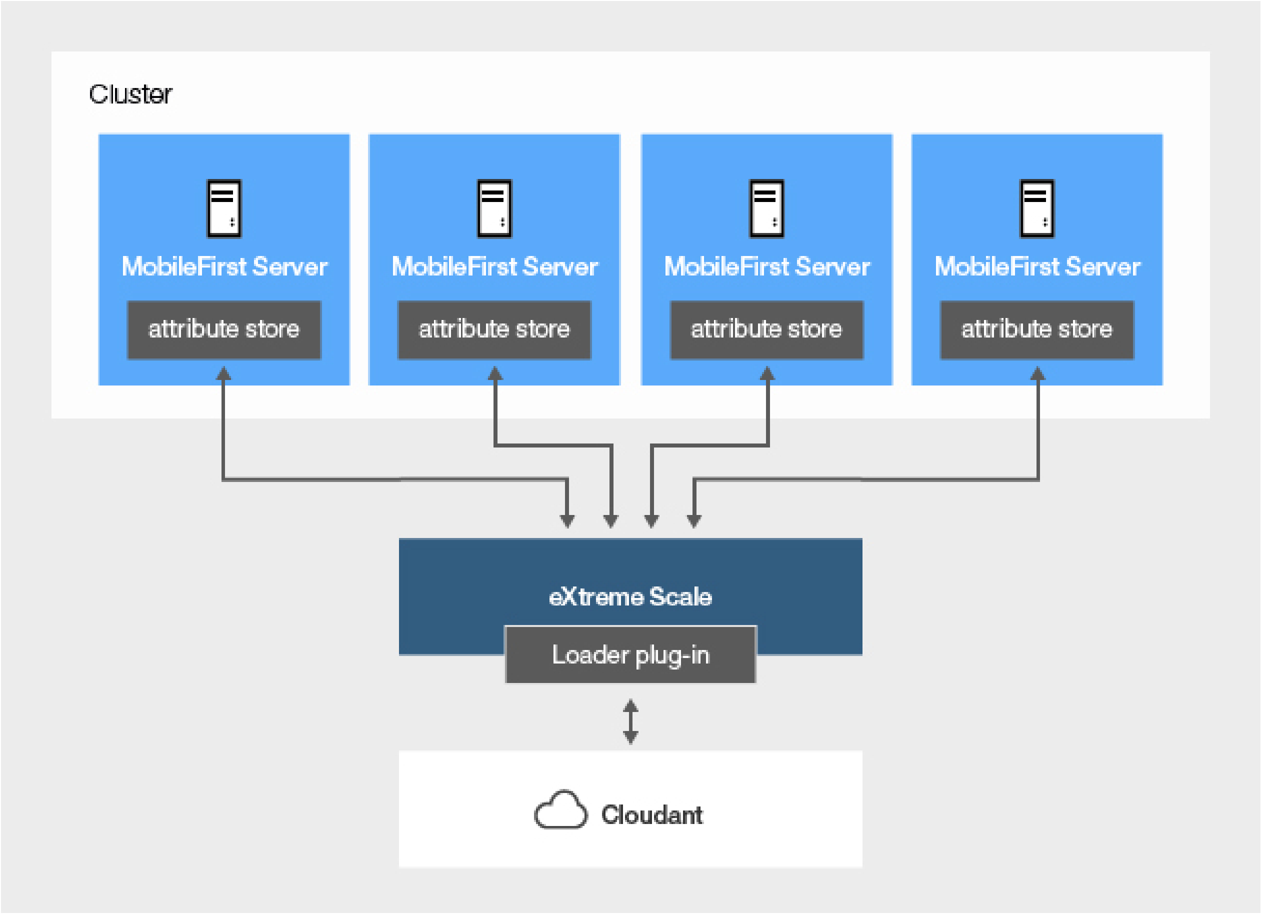


Figure 8-24. Attribute store over eXtreme Scale with Cloudant Loader plug-in

Procedure

1. Create a Cloudant account. Note the URL for the next steps.
2. Add `mfp-xs-loader.jar` to the `classpath` of your eXtreme Scale deployment.
3. In the `objectgrid.xml` file, add an entry with `className="com.worklight.core.persistence.xs.AttributeStoreCloudantLoader"` for `backingMapPluginCollection`

```
<?xml version="1.0" encoding="UTF-8"?>
<objectGridConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://ibm.com/ws/objectgrid/config ../objectGrid.xsd"
 xmlns="http://ibm.com/ws/objectgrid/config">

  <objectGrids>
    <objectGrid name="MFPGrid" txTimeout="30">
    <backingMap name="AttrStoreMap" copyMode="COPY_TO_BYTES" pluginCollectionRef="as"
        lockStrategy="PESSIMISTIC" writeBehind="T60;C100"
        nullValuesSupported="true" ttlEvictorType="LAST_ACCESS_TIME" timeToLive="10" nearCache
    </objectGrid>
  </objectGrids>
  <backingMapPluginCollections>
    <backingMapPluginCollection id="as">
        <bean id="Loader" className="com.worklight.core.persistence.xs.AttributeStoreCloudantL
    </backingMapPluginCollection>
</backingMapPluginCollections>
</objectGridConfig>
```

4. Add your Cloudant database URL to the eXtreme Scale container start script by using a JVM property, as follows:

   ```
   -Dcloudant_url= http://user:password@cloudant.com/attrstore
   ```

   where `attrstore` is a Cloudant table name. If the table does not exist, it is created automatically.
5. If Cloudant is behind a proxy, add a line for:

   ```
   -Dcloudant_proxy_host=proxy.com
   ```

   and a line for:

   ```
   -Dcloudant_proxy_port=8989
   ```

   In both lines, correct the host and port values to your proxy settings.

Client access to adapters

Clients can access adapters from three endpoints in the IBM MobileFirst Platform Server.

The following table summarizes the supported adapter types, HTTP methods, and security model for each MobileFirst Server endpoint.

Table 8-13. Client access to MobileFirst Server endpoints

Endpoint	Supported for Java adapters	Supported for JavaScript adapters	Permitted HTTP methods	Supported security model
/adapters	yes	yes	**GET, POST, PUT, DELETE, HEAD, TRACE**	OAuth 2.0-based
/apps/services/api/query	no	yes	**POST**	Pre-V7.0
/invoke	no	yes	**GET, POST**	Pre-V7.0

Accessing adapters from the `/adapters` endpoint

Clients can access both Java and JavaScript adapters from the `/adapters` endpoint on the server. You can access this endpoint from both mobile and non-mobile clients.

The URL pattern for accessing the `/adapters` endpoint is as follows:

```
http(s)://<server>:<port>/<Context>/adapters/<adapter-name>/*
```

For example, assuming that `http://mfp-server-host/project` is the IBM MobileFirst Platform Foundation for iOS project URL, and the project contains one Java adapter named `adapter1` and the adapter has a resource with path `/res1`, then `/res1` is accessible from the following URL:

```
http://mfp-server-host/project/adapters/adapter1/res1
```

Note: Using the `/adapters` endpoint is the recommended way to access IBM MobileFirst Platform Foundation for iOS adapters. This endpoint supports both JavaScript and Java adapters and is protected by the OAuth security mechanism, which was introduced beginning with MobileFirst V7.0.

Accessing adapters from a mobile client

IBM MobileFirst Platform Foundation for iOS provides a client API for accessing OAuth protected resources such as adapters. If you choose to use MobileFirst client for that purpose, it will automatically handle security for you. The following example demonstrates how to use the client API to access an adapter resource:

Native iOS client

```
NSString static *const RESOURCE_URL = @"adapters/adapter1/res1";
WLResourceRequest *request = [WLResourceRequest requestWithURL:[NSURL URLWithString:RESOURCE_URL] method:WLHttpMethodGet]
    [request sendWithCompletionHandler:^(WLResponse *response, NSError *error) {
        NSString *httpStatus = [NSString stringWithFormat:@"%d", [response status]];
        self.httpStatusTextField.text = httpStatus;
        if (error != nil) {
            [self updateView:[error description]];
        } else {
            [self updateView:[response responseText]];
        }
    }];
```

Accessing adapters from non-mobile clients

There are two ways to access an adapter from a non-mobile client:

- If you are testing adapter functionality, see Testing adapters.
- If you are accessing the adapter from a back-end system, see Exposing mobile services to non-mobile (confidential) clients.

RESTful access to JavaScript adapters

In addition to invoking JavaScript adapters over HTTP by using IBM MobileFirst Platform Foundation for iOS from the `/adapter` endpoint, you can call existing JavaScript adapter procedures over HTTP via REST URLs.

The URL pattern is as follows:

```
http(s)://<server>:<port>/<Context>/adapters/<adapter-name>/<procedure-name>
```

For example:

```
http://<hostname>:<port>/mfp/adapters/TodaysNews/getStories?params=['world']
```

Both the `GET` and `POST` methods can be used to call the adapter procedure. The procedure arguments are passed as the value of a parameter called **`params`**. This parameter is a query parameter for `GET` requests and a form parameter for `POST` requests. The value must be a JSON array of parameters that are provided in order.

Note:

For successful invocations, the status code of the HTTP response is set to 200 (OK). The response body contains the JSON output that resulted from the invocation of the JavaScript adapter. If an error occurred during adapter invocation, the status code is set to 500 (Internal Server Error).

Accessing adapters from the `/apps/services/api/query` endpoint

In versions of IBM MobileFirst Platform Foundation for iOS previous to V7.0 this endpoint was the preferred access point for clients.

The client API that can be used to access adapters from this endpoint is invokeProcedure in the WLClient class. For more information, see WL.Client iOS API.

Accessing adapters from the /invoke endpoint

JavaScript adapter procedures can be invoked by issuing an HTTP request to the MobileFirst /invoke endpoint.

The URL pattern for accessing the /invoke endpoint is as follows

```
http(s)://<server>:<port>/<Context>/invoke
```

Note: The usage of the /invoke endpoint is discouraged. Consider using the /adapters endpoint even when invoking a JavaScript adapter procedure. For more information, see "Accessing adapters from the /adapters endpoint" on page 8-118.

The following parameters are required:

Table 8-14. 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 "MobileFirst security framework" on page 8-188 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 8-252 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 the 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.

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 MobileFirst project hierarchy:

- Your own source code files and some source code files that are provided in the MobileFirst 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 MobileFirst Platform Foundation for iOS (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
|       |
|       +---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
|       |   |   |
|       |   |   +---Classes
|       |   |   |   Application Name.h
|       |   |   |   Application Name.m
|       |   |   |
|       |   |   +---CordovaLib (*)
|       |   |   +---FipsHttp
|       |   |   +---Frameworks
|       |   |   |   sqlcipher.framework (*)
|       |   |   |
|       |   |   +---Project Name Application NameIpad.xcodeproj
|       |   |   +---Resources
|       |   |   +---Settings.bundle
|       |   |   +---Tealeaf
|       |   |   +---WorklightSDK (*)
|       |   |   +---www (*)
|       |   +---nativeResources
|       +---iphone
|       |   +---css
|       |   +---images
|       |   +---js
|       |   +---native
|       |   |   |   buildtime.sh
|       |   |   |   config.xml
|       |   |   |   Entitlements-Debug.plist
|       |   |   |   Entitlements-Release.plist
|       |   |   |   main.m
|       |   |   |   Project Name Application NameIphone_Prefix.pch
|       |   |   |   Project Name Application NameIphone-Info.plist
|       |   |   |   README.txt
|       |   |   |   worklight.plist
|       |   |   |
|       |   |   +---Classes
|       |   |   |   Application Name.h
|       |   |   |   Application Name.m
|       |   |   |
|       |   |   +---CordovaLib (*)
|       |   |   +---FipsHttp
|       |   |   +---Frameworks
|       |   |   |   sqlcipher.framework (*)
|       |   |   |
|       |   |   +---Project Name Application NameIphone.xcodeproj
|       |   |   +---Resources
|       |   |   +---Settings.bundle
|       |   |   +---Tealeaf
|       |   |   +---WorklightSDK (*)
|       |   |   +---www (*)
|       |   +---nativeResources
|       +---legal
```

Note: In iOS environments, the `Frameworks` folder contains a default `.framework` file, `sqlcipher.framework`, that is automatically generated by the MobileFirst builder if it is not already in the `Frameworks` folder. The `Frameworks` folder should be committed to your source control system, but the `sqlcipher.framework` file can be ignored.

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.

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 provide resources to help you make your mobile application as accessible as possible. IBM MobileFirst Platform Foundation for iOS provides a strong foundation for building accessible applications because it supports industry standards and allows you to leverage them.

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 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.

Updating mobile apps with IBM MobileFirst Platform Foundation for iOS 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 the 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.

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 MobileFirst application is completely written by using web technologies, to accommodate new mobile operating systems supported only in later versions of IBM MobileFirst Platform Foundation for iOS. 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 (IPA) 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 bundle ID for iOS. You change the internal version of the application; for example, **`CFBundleVersion`** on iOS. For more information about application properties in the Application Center console, see “Application properties” on page 12-83.

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.

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 12-83.

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.

JSONStore

Learn about JSONStore.

JSONStore overview

JSONStore features add the ability to store JSON documents in MobileFirst applications.

JSONStore is a lightweight, document-oriented storage system that is included as a feature of IBM MobileFirst Platform Foundation for iOS, 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 native iOS applications, see the `JSONStore` Class Reference in the API reference section.

Here is a high-level summary of what JSONStore provides:

- A developer-friendly API that gives 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 MobileFirst application that contains multiple stores. For information, see “JSONStore multiple user support” on page 8-144.

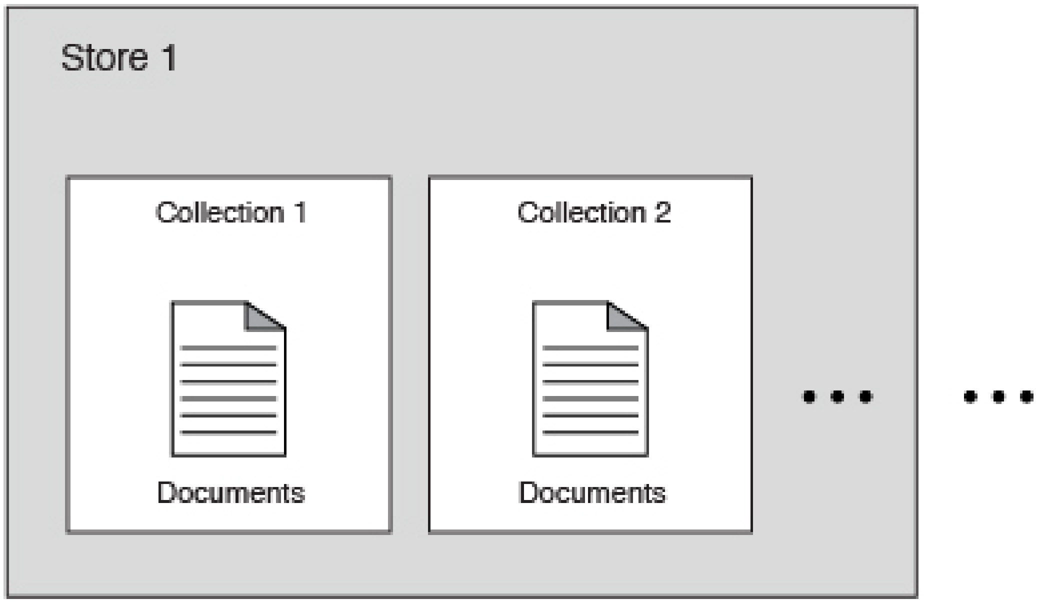


Figure 8-26. A basic graphic representation of JSONStore.

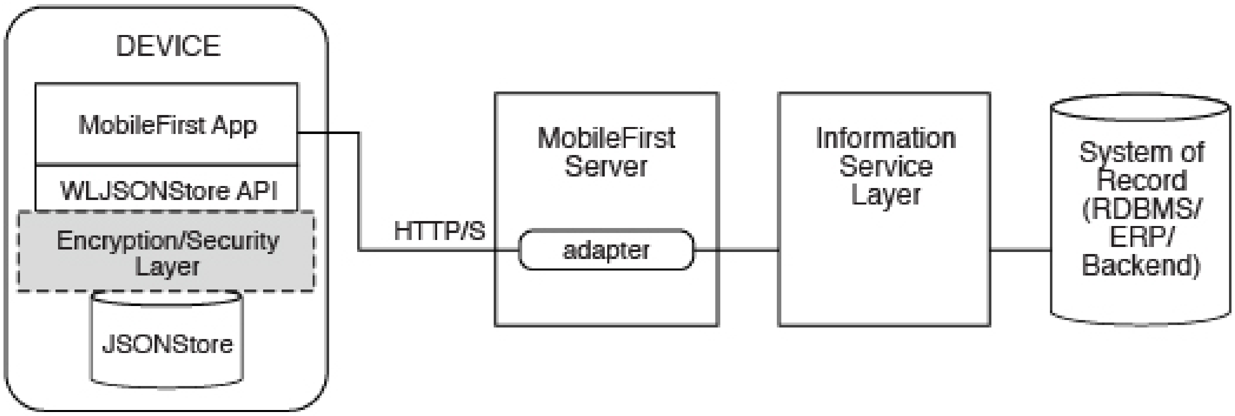

DEVICE
MobileFirst App
WLJSONStore API
Encryption/Security Layer
JSONStore
HTTP/S
MobileFirst Server
adapter
Information Service Layer
System of Record (RDBMS/ ERP/ Backend)

Figure 8-27. 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.

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 key/value pair.

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 an 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 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

If you are using IBM MobileFirst Platform Foundation for iOS without the latest interim fix installed, you can import the `JSONStore.h` header file to use the JSONStore API in a native MobileFirst iOS application. For more information about creating native MobileFirst iOS applications, see “Developing native applications” on page 8-7.

After you install the latest interim fix of IBM MobileFirst Platform Foundation for iOS, JSONStore becomes an optional feature for native MobileFirst iOS applications. To use JSONStore in a native MobileFirst iOS application, you must import the `<IBMMobileFirstPlatformFoundationJSONStore/IBMMobileFirstPlatformFoundationJSONStore.h>` umbrella header file. You can obtain the JSONStore feature in two ways:

- You can import the JSONStore framework from the worklight api folder. For more information, see “Copying SDK and configuration files from the project” on page 8-11.
- You can use CocoaPods.

  To use JSONStore in a native MobileFirst iOS application, you must enable it by adding the JSONStore SDK with CocoaPods. Your application must use CocoaPods for dependency management. Specifically,

  - You must have CocoaPods, the dependency manager for Xcode projects, installed in your development environment. For more information, see the "Getting Started" guide for CocoaPods installation.
  - Your application must be set up to use CocoaPods. For more information, see Using CocoaPods.
  - You must have an existing `Podfile`. If one does not exist, create one by using the **`pod init`** command. For more information, see Using CocoaPods.

  1. Create a `Podfile` file or edit an existing one.
     a. Create a new file that is named `Podfile` and save it in the root directory of the project.
     b. Open the `Podfile` file that is in the root directory of the project with a text editor.
     c. Add the following lines and save the file:

        ```
        source 'https://github.com/CocoaPods/Specs.git'
        pod 'IBMMobileFirstPlatformFoundationJSONStore'
        ```

     Note: The pod example assumes you are using the latest version of the MobileFirst pod. If you are are not using the latest version of MobileFirst, you need to indicate the version. For example, for importing the latest pod for 7.1.0 `IBMMobileFirstPlatformFoundation` the line would look like this:

     ```
     pod 'IBMMobileFirstPlatformFoundationJSONStore', '~> 7.1.0'
     ```

  2. Open **Terminal** and navigate to the location of the `Podfile` file.
  3. Verify that the Xcode project is closed.
  4. Type `pod install` to run the **`pod install`** command.

     This command installs the JSONStore Framework for IBM MobileFirst Platform Foundation for iOS, called the

IBMMobileFirstPlatformFoundationJSONStore.framework component, and integrates it with the mobile application Xcode project.

For more information about creating native MobileFirst iOS applications, see "Developing native applications" on page 8-7.

Note: If you are using the latest interim fix of IBM MobileFirst Platform Foundation for iOS and are upgrading your hybrid iOS application from a previous version, you must first complete the following steps.

1. Delete the existing iOS Environment (iPhone/iPad) from the hybrid project.
2. Add a new environment to the same hybrid project and select **iOS Environment (iPhone/iPad)**.

JSONStore API concepts

JSONStore provides API reference information for 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 8-144.
- All JSONStore metadata and security artifacts. For more information, see "JSONStore security" on page 8-144.

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.

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 8-133.

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 uppercase letters, the result is returned in all lowercase 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" on page 8-133.

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`).

Troubleshooting JSONStore

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, Mac OSX 10.8.3.
- JDK version. For example, Java SE Runtime Environment (build 1.7).
- IBM MobileFirst Platform Foundation for iOS version. For example, IBM Worklight V5.0.6 Developer Edition.
- iOS version. For example, iOS Simulator 6.1 or iPhone 4S iOS 6.0 (deprecated).
- Logs, such as Xcode.

Try to isolate the issue

Follow these steps to isolate the issue to more accurately report a problem.

1. Reset the simulator and call the `destroy` API to start with a clean system.
2. Ensure that you are running on a supported production environment.
   - iOS >= 6.0 simulator or device (deprecated)
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.
   - iOS simulator:

```
$ cd ~/Library/Application Support/iPhone Simulator/7.1/Applications/<id>/Documents/wljsonstore
$ sqlite3 jsonstore.sqlite
```

   - 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. 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 MobileFirst Platform Foundation for iOS 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.
- 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 8-15. 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 8-130 or "Work with external data" on page 8-146.

_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.”

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];
```

JSONStore error codes

Definitions of the error codes that are related to JSONStore.

-100 UNKNOWN_FAILURE
Unrecognized error.

-75 OS_SECURITY_FAILURE
This error code is related to the `requireOperatingSystemSecurity` flag. It can occur if the `destroy` API fails to remove security metadata that is protected by operating system security (Touch ID with passcode fallback), or the `init` or `open` APIs are unable to locate the security metadata. It can also fail if the device does not support operating system security, but operating system security usage was requested.

-50 PERSISTENT_STORE_NOT_OPEN
JSONStore is closed. Try calling the `open` method in the JSONStore class 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. See the tutorial on the Getting Started page.

-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. On Android, this error also occurs when trying to use JSONStore on unsupported architectures.

-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).

-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 `open`method with the new search fields. This error can occur if you change the name or type of the search field. For example: `{key: 'string'}` to `{key: 'number'}` or `{myKey: 'string'}` to `{theKey: 'string'}`.

-1 PERSISTENT_STORE_FAILURE
Generic Error. A malfunction in native code, most likely calling the `init` method.

0 SUCCESS
In some cases, JSONStore native code returns 0 to indicate success.

1 BAD_PARAMETER_EXPECTED_INT
Validation error.

2 BAD_PARAMETER_EXPECTED_STRING
Validation error.

3 BAD_PARAMETER_EXPECTED_FUNCTION
Validation error.

4 BAD_PARAMETER_EXPECTED_ALPHANUMERIC_STRING
Validation error.

5 BAD_PARAMETER_EXPECTED_OBJECT
Validation error.

6 BAD_PARAMETER_EXPECTED_SIMPLE_OBJECT
Validation error.

7 BAD_PARAMETER_EXPECTED_DOCUMENT
Validation error.

8 FAILED_TO_GET_UNPUSHED_DOCUMENTS_FROM_DB
The query that selects all documents that are marked dirty failed. An example in SQL of the query would be: **`SELECT * FROM [collection] WHERE _dirty > 0`**.

9 NO_ADAPTER_LINKED_TO_COLLECTION
To use functions like the `push` and `load` methods in the JSONStoreCollection 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 MobileFirst 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 JSONStoreCollection 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 MobileFirst Server because it returns a 'secure random token'. IBM 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 8-135.

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.

Objective-C API examples

You can use JSONStore for MobileFirst applications.

The following sections contain example implementations for iOS devices with JSONStore APIs. Other helpful topics include:

- "JSONStore overview" on page 8-125 - Learn about key concepts.
- "JSONStore API concepts" on page 8-130 - Learn about general information about the APIs that apply to all implementations of the JSONStore API.
- "Troubleshooting JSONStore" on page 8-134 - Learn how to debug and understand possible errors.
- "JSONStore advanced topics" on page 8-143 - 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 8-146 - Explains how to get data from an external source and send changes back to the external source.

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 MobileFirst 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 MobileFirst Adapter).

// Mark dirty documents as clean
int numCleaned = [[people markDocumentsClean:dirtyDocs error:&error] intValue];
```

Pull - get new data from a MobileFirst 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. MobileFirst 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}]
```

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 open API.

Some security artifacts (such as *salt*) are stored in the keychain. 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).

You can choose to encrypt data 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.

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.

The native iOS implementation generates 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 MobileFirst Server (requires connectivity) and trusting the server (more secure).

JSONStore multiple user support

With JSONStore, you can create multiple stores that contain different collections in a single MobileFirst application.

The `open` (Native iOS) 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) and MobileFirst application. In addition, if the employees work different shifts and handle private data from different customers while they use the MobileFirst application, multiple user support is useful.

JSONStore performance

Learn about the factors that can affect JSONStore performance.

Network

- Ideally, getting and sending data from and to a MobileFirst 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 MobileFirst 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 MobileFirst 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

- 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.
- Allow Automatic Reference Counting to work, but do not depend on it entirely.

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 `open` APIs and other operations that work with documents inside the collection. Consider whether security is genuinely required. For example, the `open` API is much slower with encryption because it must generate the encryption keys that are used 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.

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).

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 8-140.

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 MobileFirst 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 MobileFirst 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.

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 MobileFirst adapters for the Transport Layer. Some of the advantages of using MobileFirst 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: MobileFirst adapter

You can choose to send dirty documents to a MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 security utilities

Learn about JSONStore security utilities.

JSONStore security utilities overview

The MobileFirst 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 MobileFirst 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"
```

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:NSUTF8StringEncoding];
```

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];
```

Storing mobile data in Cloudant

You can store data for your mobile application in a Cloudant database. Cloudant is an advanced NoSQL database that can handle a wide variety of data types, such as JSON, full-text, and geospatial data. The SDK is available for Objective-C and Swift.

CloudantToolkit and IMFData frameworks are deprecated

Use the CDTDatastore SDK as a replacement for CloudantToolkit and IMFData frameworks.

If you want to access remote stores directly, use REST calls in your application and refer to the Cloudant API Reference.

Cloudant versus JSONStore

You might consider using JSONStore instead of Cloudant in the following scenarios:

- When you are storing data on the mobile device that must be stored in a FIPS 140-2 compliant manner.
- When you need to synchronize data between the device and the enterprise.
- When you are developing a hybrid application.

For more information about JSONStore, see “JSONStore” on page 8-125.

Integrating MobileFirst and Cloudant security

Adapter sample

To download the sample, see Sample: mfp-bluelist-on-premises.

To understand the MobileFirst adapter that is included with the Bluelist sample, you must understand both Cloudant security and “MobileFirst security framework” on page 8-188.

The Bluelist adapter sample has two primary functions:

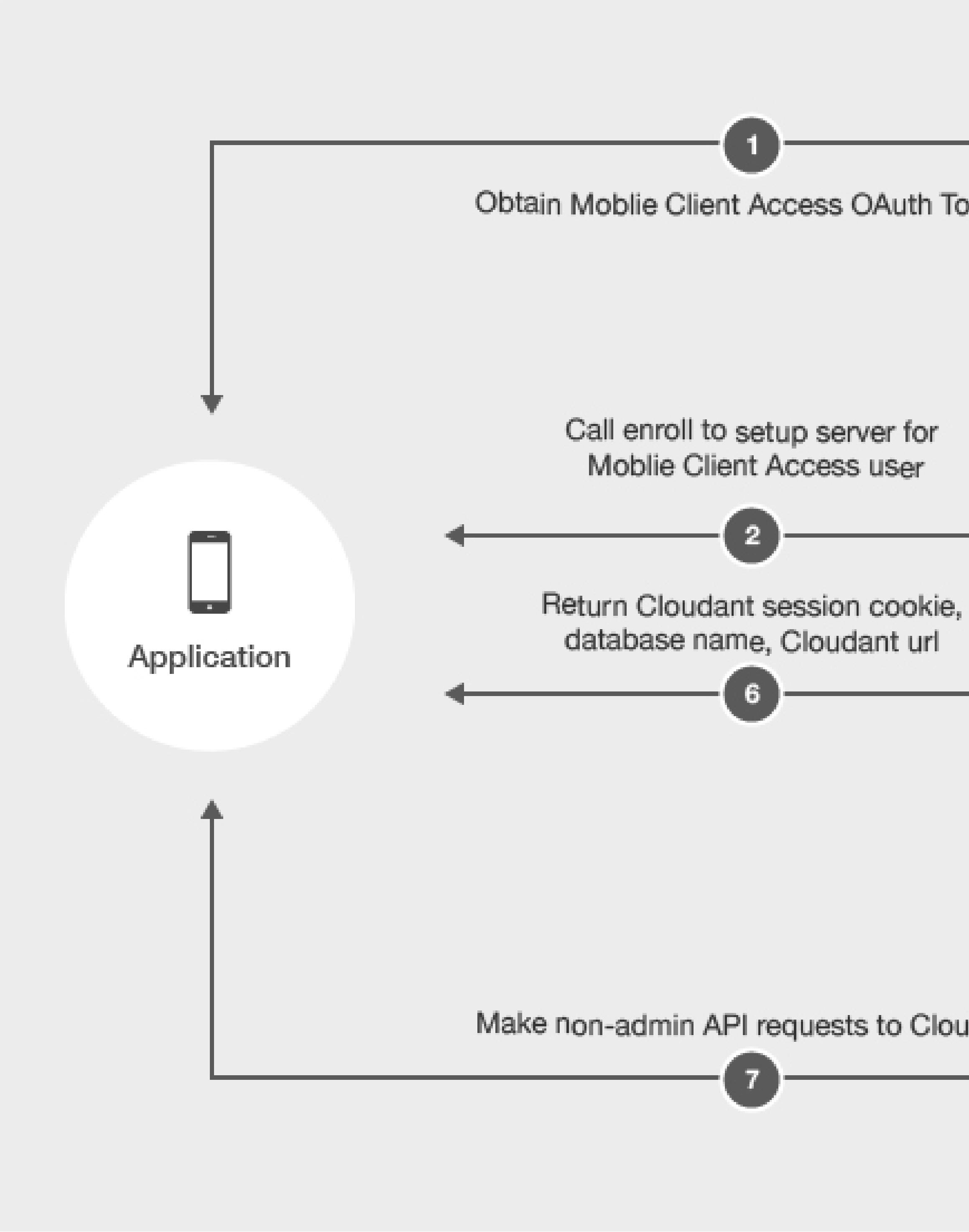
- Exchange MobileFirst OAuth tokens for Cloudant session cookies
- Perform the required `admin` requests to Cloudant from the Bluelist sample.

The sample demonstrates how to perform API requests that require `admin` access on the server where it is secure. While it is possible to place your admin credentials on the mobile device, it is a better practice to restrict access from mobile devices.

The Bluelist sample integrates MobileFirst security with Cloudant security. The MobileFirst adapter sample maps a MobileFirst identity to a Cloudant identity. The mobile device receives a Cloudant session cookie to perform non-admin API requests. The sample uses the Couch Security model.

Enroll REST endpoint

The following diagram illustrates the integration performed by the Bluelist adapter sample `/enroll` endpoint.


1
Obtain Moblie Client Access OAuth To
Call enroll to setup server for
Moblie Client Access user
2
Return Cloudant session cookie,
database name, Cloudant url
6
Application
Make non-admin API requests to Clou
7

1. Mobile device obtains the MobileFirst OAuth token from the MobileFirst Server.
2. Mobile device calls the `/enroll` endpoint on the MobileFirst adapter.
3. The MobileFirst adapter sample validates the MobileFirst OAuth token with the MobileFirst Server.

4. If valid, performs admin API requests to Cloudant. The sample checks for an existing Cloudant user in the _users database.
   - If the user exists, look up Cloudant user credentials in the _users database.
   - If a new user is passed, use the Cloudant admin credentials, create a new Cloudant user and store in the _users database.
   - Generate a unique database name for the user and create a remote database on Cloudant with that name.
   - Give the Cloudant user permissions to read/write the newly created database.
   - Create the required indexes for the Bluelist application.
5. Request a new Cloudant session cookie.
6. The MobileFirst adapter sample returns a Cloudant session cookie, remote database name, and Cloudant URL to the mobile device.
7. Mobile device makes requests directly to Cloudant until the session cookie expires.

sessioncookie REST Endpoint

In the case of an expired session cookie, the mobile device can exchange a valid MobileFirst OAuth token for a Cloudant session cookie with the /sessioncookie endpoint.

Creating databases

Accessing local data stores

You can use a local data store to store data on the client device for fast access, even when offline.

Procedure

To create Store objects to access a local database, supply a name for the data store.

Important: The database name must be in lowercase.
BEFORE (with IMFData/CloudantToolkit):

```
//Get reference to data manager
IMFDataManager *manager = [IMFDataManager sharedInstance];
NSString *name = @"automobiledb";
NSError *error = nil;

//Create local store
CDTStore *store = [manager localStore:name error:&error];
```

```
let manager = IMFDataManager.sharedInstance()
let name = "automobiledb"

var store:CDTStore?
do {
    store = try manager.localStore(name)
} catch let error as NSError {
    // Handle error
}
```

AFTER (with Cloudant Sync):

```
// Get reference to datastore manager
CDTDatastoreManager *datastoreManager = existingDatastoreManager;
NSString *name = @"automobiledb";
NSError *error = nil;
```

```
//Create datastore
CDTDatastore *datastore = [datastoreManager datastoreNamed:name error:&error];
```

```
// Get reference to datastore manager
let datastoreManager:CDTDatastoreManager = existingDatastoreManager
let name:String  = "automobiledb"

//Create local store
var datastore:CDTDatastore?
do{
    datastore = try datastoreManager.datastoreNamed(name)
}catch let error as NSError{
    // Handle error
}
```

Creating remote data stores

Procedure

To save data in the remote store, supply the data store name.
BEFORE (with IMFData/CloudantToolkit):

```
// Get reference to data manager
IMFDataManager *manager = [IMFDataManager sharedInstance];
NSString *name = @"automobiledb";

// Create remote store
[manager remoteStore:name completionHandler:^(CDTStore *createdStore, NSError *error) {
    if(error){
        // Handle error
    }else{
        CDTStore *store = createdStore;
        NSLog(@"Successfully created store: %@", store.name);
    }
}];
```

```
let manager = IMFDataManager.sharedInstance()
let name = "automobiledb"

manager.remoteStore(name, completionHandler: { (createdStore:CDTStore!, error:NSError!) -> Void ir
    if nil != error {
        //Handle error
    } else {
        let store:CDTStore = createdStore
        print("Successfully created store: \(store.name)")
    }
})
```

AFTER (with Cloudant Sync):
See Cloudant REST documentation for Database Create.

Encrypting data on the device

To enable the encryption of local data stores on mobile devices, you must make updates to your application to include encryption capabilities and create encrypted data stores.

Encrypting data on iOS devices

Procedure

1. Obtain the encryption capabilities with CocoaPods.
   a. Open your `Podfile` and add the following line:
      BEFORE (with IMFData/CloudantToolkit):

```
pod 'IMFDataLocal/SQLCipher'
```

AFTER (with Cloudant Sync):

```
pod 'CDTDatastore/SQLCipher'
```

For more information, see the CDTDatastore encryption documentation.

b. Run the following command to add the dependencies to your application.

```
pod install
```

2. To use the encryption feature within a Swift application, add the following imports to the associated bridging header for the application: BEFORE (with IMFData/CloudantToolkit):

```
#import <CloudantSync.h>
#import <CloudantSyncEncryption.h>
#import <CloudantToolkit/CloudantToolkit.h>
#import <IMFData/IMFData.h>
```

AFTER (with Cloudant Sync):

```
#import <CloudantSync.h>
#import <CloudantSyncEncryption.h>
```

3. Initialize your local store for encryption with a key provider.

Note: If you change the password after creating the database, an error occurs because the existing database cannot be decrypted. You cannot change your password after the database has been encrypted. You must delete the database to change passwords.

BEFORE (with IMFData/CloudantToolkit):

```
//Get reference to data manager
IMFDataManager *manager = [IMFDataManager sharedInstance];
NSString *name = @"automobiledb";
NSError *error = nil;

// Initalize a key provider
id<CDTEncryptionKeyProvider> keyProvider = [CDTEncryptionKeychainProvider providerWithPassword: @

//Initialize local store
CDTStore *localStore = [manager localStore: name withEncryptionKeyProvider: keyProvider error: &e
```

```
let manager = IMFDataManager.sharedInstance()
let name = "automobiledb"

let keyProvider = CDTEncryptionKeychainProvider(password: "passw0rd", forIdentifier: "identifier"
var store:CDTStore?
do {
    store = try manager.localStore(name, withEncryptionKeyProvider: keyProvider)
} catch let error as NSError {
    // Handle error
}
```

AFTER (with Cloudant Sync):

```
// Get reference to datastore manager
CDTDatastoreManager *datastoreManager = existingDatastoreManager;
NSString *name = @"automobiledb";
NSError *error = nil;

// Create KeyProvider
id<CDTEncryptionKeyProvider> keyProvider = [CDTEncryptionKeychainProvider providerWithPassword: @

//Create local store
CDTDatastore *datastore = [datastoreManager datastoreNamed:name withEncryptionKeyProvider:keyProv
```

```
// Get reference to datastore manager
let datastoreManager:CDTDatastoreManager = existingDatastoreManager
let name:String  = "automobiledb"

//Create local store
var datastore:CDTDatastore?
let keyProvider = CDTEncryptionKeychainProvider(password: "passw0rd", forIdentifier: "identifi
do{
    datastore = try datastoreManager.datastoreNamed(name, withEncryptionKeyProvider: keyProvid
}catch let error as NSError{
    // Handle error
}
```

4. When you are replicating data with an encrypted local store, you must initialize the CDTPullReplication and CDTPushReplication methods with a key provider.

   BEFORE (with IMFData/CloudantToolkit):

```
//Get reference to data manager
IMFDataManager *manager = [IMFDataManager sharedInstance];
NSString *databaseName = @"automobiledb";

// Initalize a key provider
id<CDTEncryptionKeyProvider> keyProvider = [CDTEncryptionKeychainProvider providerWithPassword

// pull replication
CDTPullReplication *pull = [manager pullReplicationForStore: databaseName withEncryptionKeyPro


// push replication
CDTPushReplication *push = [manager pushReplicationForStore: databaseName withEncryptionKeyPro
```

```
//Get reference to data manager
let manager = IMFDataManager.sharedInstance()
let databaseName = "automobiledb"

// Initalize a key provider
let keyProvider = CDTEncryptionKeychainProvider(password: "password", forIdentifier: "identifi

// pull replication
let pull:CDTPullReplication = manager.pullReplicationForStore(databaseName, withEncryptionKeyP


// push replication
let push:CDTPushReplication = manager.pushReplicationForStore(databaseName, withEncryptionKeyP
```

   AFTER (with Cloudant Sync):

   Replication with an encrypted database requires no changes from replication with an unencrypted database.

Setting user permissions

You can set user permissions on remote databases.

Procedure

Set user permissions on the remote store.
BEFORE (with IMFData/CloudantToolkit):

```
// Get reference to data manager
IMFDataManager *manager = [IMFDataManager sharedInstance];

// Set permissions for current user on a store
[manager setCurrentUserPermissions: DB_ACCESS_GROUP_MEMBERS forStoreName: @"automobiledb" completionHander:^(BOOL success, NSError *error) {
    if(error){
        // Handle error
    }else{
        // setting permissions was successful
    }
}];
```

```
// Get reference to data manager
let manager = IMFDataManager.sharedInstance()

// Set permissions for current user on a store
manager.setCurrentUserPermissions(DB_ACCESS_GROUP_MEMBERS, forStoreName: "automobiledb") { (success:Bool, error:NSError!) -> Void in
    if nil != error {
        // Handle error
    } else {
        // setting permissions was successful
    }
}
```

AFTER (with Cloudant Sync): You cannot set user permissions from the mobile device. You must set permissions with the Cloudant dashboard or server-side code. For a sample of how to integrate MobileFirst OAuth tokens with Cloudant Security, see the Bluelist sample.

Modeling data

Cloudant stores data as JSON documents. To store data as objects in your application, use the included data object mapper class that maps native objects to the underlying JSON document format.

About this task

Cloudant stores data as JSON documents. The CloudantToolkit framework provided an object mapper to map between native objects and JSON documents. The CDTDatastore API does not provide this feature. The snippets in the following sections demonstrate how to use CDTDatastore objects to accomplish the same operations.

Performing CRUD operations

You can modify the content of a data store.

About this task

For more details on create, retrieve, update, and delete (CRUD) operations, see CDTDatastore CRUD documentation.

For create, retrieve, update, and delete (CRUD) operations on a remote store, see the Cloudant Document API

Creating data

Procedure

Save data.
BEFORE (with IMFData/CloudantToolkit):

```
// Use an existing store
CDTStore *store = existingStore;

// Create your Automobile to save
Automobile *automobile = [[Automobile alloc] initWithMake:@"Toyota" model:@"Corolla" year: 2006];

[store save:automobile completionHandler:^(id savedObject, NSError *error) {
    if (error) {
        // save was not successful, handler received an error
    } else {
        // use the result
        Automobile *savedAutomobile = savedObject;
        NSLog(@"saved revision: %@", savedAutomobile);
    }
}];
```

```
 // Use an existing store
let store:CDTStore = existingStore

// Create your object to save
let automobile = Automobile(make: "Toyota", model: "Corolla", year: 2006)

store.save(automobile, completionHandler: { (savedObject:AnyObject!, error:NSError!) -> Void in
    if nil != error {
        //Save was not successful, handler received an error
    } else {
        // Use the result
        print("Saved revision: \(savedObject)")
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store
CDTDatastore *datastore = existingDatastore;

// Create document body
CDTMutableDocumentRevision * revision = [CDTMutableDocumentRevision revision];
revision.body = @{@"@datatype" : @"Automobile", @"make" :@"Toyota", @"model": @"Corolla", @"year" : @2006};

NSError *error = nil;
CDTDocumentRevision *createdRevision = [datastore createDocumentFromRevision:revision error:&error];

if (error) {
    // save was not successful, handler received an error
} else {
    // use the result
    NSLog(@"Revision: %@", createdRevision);
}
```

```
// Use an existing store
let datastore:CDTDatastore = existingDatastore

// Create document body
let revision = CDTMutableDocumentRevision()
revision.setBody(["make":"Toyota","model":"Corolla","year":2006])

var createdRevision:CDTDocumentRevision?
do{
    createdRevision = try datastore.createDocumentFromRevision(revision)
    NSLog("Revision: \(createdRevision)");
}catch let error as NSError{
    // Handle error
}
```

Reading data

You can fetch data.

Procedure

Read data.
BEFORE (with IMFData/CloudantToolkit):

```
CDTStore *store = existingStore;
NSString *automobileId = existingAutomobileId;

// Fetch Autombile from Store
```

```
[store fetchById:automobileId completionHandler:^(id object, NSError *error) {
    if (error) {
        // fetch was not successful, handler received an error
    } else {
        // use the result
        Automobile *savedAutomobile = object;
        NSLog(@"fetched automobile: %@", savedAutomobile);
    }
}];
```

```
// Using an existing store and Automobile
let store:CDTStore = existingStore
let automobileId:String = existingAutomobileId

// Fetch Autombile from Store
store.fetchById(automobileId, completionHandler: { (object:AnyObject!, error:NSError!) -> Void in
    if nil != error {
        // Fetch was not successful, handler received an error
    } else {
        // Use the result
        let savedAutomobile:Automobile = object as! Automobile
        print("Fetched automobile: \(savedAutomobile)")
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store and documentId
CDTDatastore *datastore = existingDatastore;
NSString *documentId = existingDocumentId;

// Fetch the CDTDocumentRevision from the store
NSError *error = nil;
CDTDocumentRevision *fetchedRevision = [datastore getDocumentWithId:documentId error:&error];

if (error) {
    // fetch was not successful, handler received an error
} else {
    // use the result
    NSLog(@"Revision: %@", fetchedRevision);
}
```

```
// Use an existing store and documentId
let datastore:CDTDatastore = existingDatastore
let documentId:String = existingDocumentId

var fetchedRevision:CDTDocumentRevision?
do{
    fetchedRevision = try datastore.getDocumentWithId(documentId)
    NSLog("Revision: \(fetchedRevision)");
}catch let error as NSError{
    // Handle error
}
```

Updating data

To update an object, run a save on an existing object. Because the item exists, it is updated.

Procedure

Update objects.
BEFORE (with IMFData/CloudantToolkit):

```
// Use an existing store and Automobile
CDTStore *store = existingStore;
Automobile *automobile = existingAutomobile;
```

```
// Update some of the values in the Automobile
automobile.year = 2015;

// Save Autombile to the store
[store save:automobile completionHandler:^(id savedObject, NSError *error) {
    if (error) {
        // sasve was not successful, handler received an error
    } else {
        // use the result
        Automobile *savedAutomobile = savedObject;
        NSLog(@"saved automobile: %@", savedAutomobile);
    }
}];
```

```
// Use an existing store and Automobile
let store:CDTStore = existingStore
let automobile:Automobile = existingAutomobile

// Update some of the values in the Automobile
automobile.year = 2015

// Save Autombile to the store
store.save(automobile, completionHandler: { (savedObject:AnyObject!, error:NSError!) -> Void in
    if nil != error {
        // Update was not successful, handler received an error
    } else {
        // Use the result
        let savedAutomobile:Automobile = savedObject as! Automobile
        print("Updated automobile: \(savedAutomobile)")
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store and document
CDTDatastore *datastore = existingDatastore;
CDTMutableDocumentRevision *documentRevision = [existingDocumentRevision mutableCopy];

// Update some of the values in the revision
[documentRevision.body setValue:@2015 forKey:@"year"];

NSError *error = nil;
CDTDocumentRevision *updatedRevision = [datastore updateDocumentFromRevision:documentRevision error:&error];
if (error) {
    // save was not successful, handler received an error
} else {
    // use the result
    NSLog(@"Revision: %@", updatedRevision);
}
```

```
// Use an existing store and document
let datastore:CDTDatastore = existingDatastore
let documentRevision:CDTMutableDocumentRevision = existingDocumentRevision.mutableCopy()

// Update some of the values in the revision
documentRevision.body()["year"] = 2015

var updatedRevision:CDTDocumentRevision?
do{
    updatedRevision = try datastore.updateDocumentFromRevision(documentRevision)
    NSLog("Revision: \(updatedRevision)");
}catch let error as NSError{
    // Handle error
}
```

Deleting data

To delete an object, pass the object that you want to delete to the store.

Procedure

Delete objects.
BEFORE (with IMFData/CloudantToolkit):

```
// Using an existing store and Automobile
CDTStore *store = existingStore;
Automobile *automobile = existingAutomobile;

// Delete the Automobile object from the store
[store delete:automobile completionHandler:^(NSString *deletedObjectId, NSString *deletedRevisionId, NSError *error) {
    if (error) {
        // delete was not successful, handler received an error
    } else {
        // use the result
        NSLog(@"deleted Automobile doc-%@-rev-%@", deletedObjectId, deletedRevisionId);
    }
}];
```

```
// Using an existing store and Automobile
let store:CDTStore = existingStore
let automobile:Automobile = existingAutomobile

// Delete the Automobile object
store.delete(automobile, completionHandler: { (deletedObjectId:String!, deletedRevisionId:String!, error:NSError!) -> Void
    if nil != error {
        // delete was not successful, handler received an error
    } else {
        // use the result
        print("deleted document doc-\(deletedObjectId)-rev-\(deletedRevisionId)")
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store and revision
CDTDatastore *datastore = existingDatastore;
CDTDocumentRevision *documentRevision = existingDocumentRevision;

// Delete the CDTDocumentRevision from the store
NSError *error = nil;
CDTDocumentRevision *deletedRevision = [datastore deleteDocumentFromRevision:documentRevision error:&error];
if (error) {
    // delete was not successful, handler received an error
} else {
    // use the result
    NSLog(@"deleted document: %@", deletedRevision);
}
```

```
// Use an existing store and revision
let datastore:CDTDatastore = existingDatastore
let documentRevision:CDTDocumentRevision = existingDocumentRevision

var deletedRevision:CDTDocumentRevision?
do{
    deletedRevision = try datastore.deleteDocumentFromRevision(documentRevision)
    NSLog("Revision: \(deletedRevision)");
}catch let error as NSError{
    // Handle error
}
```

Creating indexes

To perform queries, you must create an index.

About this task

For more details, see CDTDatastore Query documentation. For query operations on a remote store, see the Cloudant Query API.

Procedure

- Create an index that includes the data type. Indexing with the data type is useful when an object mapper is set on the data store.

  BEFORE (with IMFData/CloudantToolkit):

```
 // Use an existing data store
CDTStore *store = existingStore;

// The data type to use for the Automobile class
NSString *dataType = [store.mapper dataTypeForClassName:NSStringFromClass([Automobile class])];

// Create the index
[store createIndexWithDataType:dataType fields:@[@"year", @"make"] completionHandler:^(NSError *error) {
    if(error){
        // Handle error
    }else{
        // Continue application flow
    }
}];
```

```
// A store that has been previously created.
let store:CDTStore = existingStore

// The data type to use for the Automobile class
let dataType:String = store.mapper.dataTypeForClassName(NSStringFromClass(Automobile.classForCoder()))

// Create the index
store.createIndexWithDataType(dataType, fields: ["year","make"]) { (error:NSError!) -> Void in
    if nil != error {
        // Handle error
    } else {
        // Continue application flow
    }
}
```

  AFTER (with Cloudant Sync):

```
// A store that has been previously created.
CDTDatastore *datastore = existingDatastore;

NSString *indexName = [datastore ensureIndexed:@[@"@datatype", @"year", @"make"] withName:@"automobileindex"];
if(!indexName){
    // Handle error
}
```

```
// A store that has been previously created.
let datastore:CDTDatastore = existingDatastore

// Create the index
let indexName:String? = datastore.ensureIndexed(["@datatype","year","make"], withName: "automobileindex")
if(indexName == nil){
    // Handle error
}
```

- Delete indexes.

  BEFORE (with IMFData/CloudantToolkit):

```
// Use an existing data store
CDTStore *store = existingStore;
NSString *indexName = existingIndexName;

// Delete the index
[store deleteIndexWithName:indexName completionHandler:^(NSError *error) {
    if(error){
        // Handle error
    }else{
        // Continue application flow
    }
}];
```

```
// Use an existing store
let store:CDTStore = existingStore

// The data type to use for the Automobile class
let dataType:String = store.mapper.dataTypeForClassName(NSStringFromClass(Automobile.classForCoder()))

// Delete the index
store.deleteIndexWithDataType(dataType, completionHandler: { (error:NSError!) -> Void in
    if nil != error {
        // Handle error
    } else {
        // Continue application flow
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store
CDTDatastore *datastore = existingDatastore;
NSString *indexName = existingIndexName;

// Delete the index
BOOL success = [datastore deleteIndexNamed:indexName];
if(!success){
    // Handle error
}
```

```
// A store that has been previously created.
let datastore:CDTDatastore = existingDatastore
let indexName:String = existingIndexName

// Delete the index
let success:Bool = datastore.deleteIndexNamed(indexName)
if(!success){
    // Handle error
}
```

Querying data

After you create an index, you can query the data in your database.

About this task

For more details, see CDTDatastore Query documentation.

For query operations on a remote store, see the Cloudant Query API.

Procedure

Create and run a query on iOS.
BEFORE (with IMFData/CloudantToolkit):

```
// Use an existing store
CDTStore *store = existingStore;

NSPredicate *queryPredicate = [NSPredicate predicateWithFormat:@"(year = 2006)"];
CDTCloudantQuery *query = [[CDTCloudantQuery alloc]
initDataType:[store.mapper dataTypeForClassName:NSStringFromClass([Automobile class])] withPredicate:queryPredicate];

[store performQuery:query completionHandler:^(NSArray *results, NSError *error) {
    if(error){
        // Handle error
    }else{
        // Use result of query.  Result will be Automobile objects.
    }
}];
```

```
// Use an existing store
let store:CDTStore = existingStore

let queryPredicate:NSPredicate = NSPredicate(format:"(year = 2006)")
let query:CDTCloudantQuery = CDTCloudantQuery(dataType: "Automobile", withPredicate: queryPredicate)

store.performQuery(query, completionHandler: { (results:[AnyObject]!, error:NSError!) -> Void in
    if nil != error {
        // Handle error
    } else {
        // Use result of query.  Result will be Automobile objects.
    }
})
```

AFTER (with Cloudant Sync):

```
// Use an existing store
CDTDatastore *datastore = existingDatastore;

CDTQResultSet *results = [datastore find:@{@"@datatype" : @"Automobile", @"year" : @2006}];
if(results){
    // Use results
}
```

```
// Use an existing store
let datastore:CDTDatastore = existingDatastore

let results:CDTQResultSet? = datastore.find(["@datatype" : "Automobile", "year" : 2006])
if(results == nil){
    // Handle error
}
```

Supporting offline storage and synchronization

You can synchronize the data on a mobile device with a remote database instance. You can either pull updates from a remote database to the local database on the mobile device, or push local database updates to a remote database.

Before you begin

For more details, see CDTDatastore Replication documentation.

For CRUD operations on a remote store, see the Cloudant Replication API

Running pull replication

Procedure

Run pull replication.
BEFORE (with IMFData/CloudantToolkit):

```
// store is an existing CDTStore object created using IMFDataManager remoteStore
__block NSError *replicationError;
CDTPullReplication *pull = [manager pullReplicationForStore: store.name];
CDTReplicator *replicator = [manager.replicatorFactory oneWay:pull error:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator creation was successful
}
```

```
[replicator startWithError:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator start was successful
}

// (optionally) monitor replication via polling
while (replicator.isActive) {
    [NSThread sleepForTimeInterval:1.0f];
    NSLog(@"replicator state : %@", [CDTReplicator stringForReplicatorState:replicator.state]);
}
```

```
// Use an existing store
let store:CDTStore = existingStore

do {
    // store is an existing CDTStore object created using IMFDataManager remoteStore
    let pull:CDTPullReplication = manager.pullReplicationForStore(store.name)
    let replicator:CDTReplicator = try manager.replicatorFactory.oneWay(pull)

    // start replication
    try replicator.start()

    // (optionally) monitor replication via polling
    while replicator.isActive() {
        NSThread.sleepForTimeInterval(1.0)
        print("replicator state : \(CDTReplicator.stringForReplicatorState(replicator.state))")
    }

} catch let error as NSError {
    // Handle error
}
```

AFTER (with Cloudant Sync):

```
// Use an existing datastore
NSURL *remoteStoreUrl = existingRemoteStoreUrl;
CDTDatastoreManager *datastoreManager = existingDatastoreManager;
CDTDatastore *datastore = existingDatastore;

// Create pull replication objects
__block NSError *replicationError;
CDTReplicatorFactory *replicatorFactory = [[CDTReplicatorFactory alloc]initWithDatastoreManager:datastoreManager];
CDTPullReplication *pull = [CDTPullReplication replicationWithSource:remoteStoreUrl target:datastore];
CDTReplicator *replicator = [replicatorFactory oneWay:pull error:&error];
if(replicationError){
    // Handle error
}else{
    // replicator creation was successful
}

[replicator startWithError:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator start was successful
}

// (optionally) monitor replication via polling
while (replicator.isActive) {
    [NSThread sleepForTimeInterval:1.0f];
    NSLog(@"replicator state : %@", [CDTReplicator stringForReplicatorState:replicator.state]);
}
```

```
let remoteStoreUrl:NSURL = existingRemoteStoreUrl
let datastoreManager:CDTDatastoreManager = existingDatastoreManager
let datastore:CDTDatastore = existingDatastore
```

```
do {
    // store is an existing CDTStore object created using IMFDataManager remoteStore
    let replicatorFactory = CDTReplicatorFactory(datastoreManager: datastoreManager)
    let pull:CDTPullReplication = CDTPullReplication(source: remoteStoreUrl, target: datastore)
    let replicator:CDTReplicator = try replicatorFactory.oneWay(pull)

    // start replication
    try replicator.start()

    // (optionally) monitor replication via polling
    while replicator.isActive() {
        NSThread.sleepForTimeInterval(1.0)
        print("replicator state : \(CDTReplicator.stringForReplicatorState(replicator.state))")
    }

} catch let error as NSError {
    // Handle error
}
```

Running push replication

Procedure

Run push replication.
BEFORE (with IMFData/CloudantToolkit):

```
// store is an existing CDTStore object created using IMFDataManager localStore
__block NSError *replicationError;
CDTPushReplication *push = [manager pushReplicationForStore: store.name];
CDTReplicator *replicator = [manager.replicatorFactory oneWay:push error:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator creation was successful
}

[replicator startWithError:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator start was successful
}

// (optionally) monitor replication via polling
while (replicator.isActive) {
    [NSThread sleepForTimeInterval:1.0f];
    NSLog(@"replicator state : %@", [CDTReplicator stringForReplicatorState:replicator.state]);
}
```

```
// Use an existing store
let store:CDTStore = existingStore

do {
    // store is an existing CDTStore object created using IMFDataManager localStore
    let push:CDTPushReplication = manager.pushReplicationForStore(store.name)
    let replicator:CDTReplicator = try manager.replicatorFactory.oneWay(push)

    // Start replication
    try replicator.start()

    // (optionally) monitor replication via polling
    while replicator.isActive() {
        NSThread.sleepForTimeInterval(1.0)
        print("replicator state : \(CDTReplicator.stringForReplicatorState(replicator.state))")
    }
} catch let error as NSError {
    // Handle error
}
```

AFTER (with Cloudant Sync):

```
// Use an existing datastore
NSURL *remoteStoreUrl = existingRemoteStoreUrl;
CDTDatastoreManager *datastoreManager = existingDatastoreManager;
CDTDatastore *datastore = existingDatastore;

// Create push replication objects
__block NSError *replicationError;
CDTReplicatorFactory *replicatorFactory = [[CDTReplicatorFactory alloc]initWithDatastoreManager:datastoreManager];
CDTPushReplication *push = [CDTPushReplication replicationWithSource:datastore target:remoteStoreUrl];
CDTReplicator *replicator = [replicatorFactory oneWay:push error:&error];
if(replicationError){
    // Handle error
}else{
    // replicator creation was successful
}

[replicator startWithError:&replicationError];
if(replicationError){
    // Handle error
}else{
    // replicator start was successful
}

// (optionally) monitor replication via polling
while (replicator.isActive) {
    [NSThread sleepForTimeInterval:1.0f];
    NSLog(@"replicator state : %@", [CDTReplicator stringForReplicatorState:replicator.state]);
}
```

```
let remoteStoreUrl:NSURL = existingRemoteStoreUrl
let datastoreManager:CDTDatastoreManager = existingDatastoreManager
let datastore:CDTDatastore = existingDatastore


do {
    // store is an existing CDTStore object created using IMFDataManager remoteStore
    let replicatorFactory = CDTReplicatorFactory(datastoreManager: datastoreManager)
    let push:CDTPushReplication = CDTPushReplication(source: datastore, target: remoteStoreUrl)
    let replicator:CDTReplicator = try replicatorFactory.oneWay(push)

    // start replication
    try replicator.start()

    // (optionally) monitor replication via polling
    while replicator.isActive() {
        NSThread.sleepForTimeInterval(1.0)
        print("replicator state : \(CDTReplicator.stringForReplicatorState(replicator.state))")
    }

} catch let error as NSError {
    // Handle error
}
```

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 MobileFirst 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 MobileFirst Server to specialized Apple servers, and from there to the relevant phones. The unified push notification mechanism in IBM MobileFirst Platform Foundation for iOS makes the entire process of communicating with the users and devices completely transparent to the developer.

The following diagram shows an example of a push notification mechanism where notifications are sent from the MobileFirst Server to specialized servers or gateways and from there to the relevant phones.

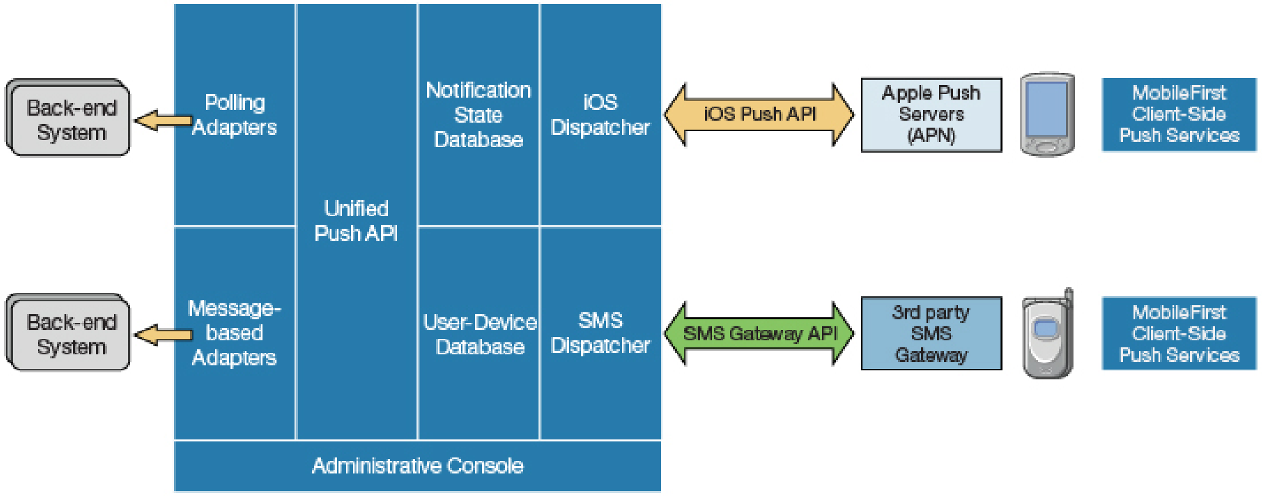


Figure 8-28. Push notification mechanism

iOS apps use the Apple Push Notification Service (APNS). For more information about setting up push notification, see “Setting up push notifications” on page 8-172.

Proxy settings

Use the proxy settings to set the optional proxy through which notifications are sent to APNS. You can set the proxy by using the **`push.apns.proxy.*`** configuration properties. For more information, see “Configuration of MobileFirst applications on the server” on page 11-50.

Architecture

Unlike other IBM MobileFirst Platform Foundation for iOS services, the push server requires outbound connections to Apple server using port that is defined by Apple.

For more information, see "Possible MobileFirst push notification architectures."

Possible MobileFirst push notification architectures

IBM MobileFirst Platform Foundation for iOS supports two different methods of implementing push notifications, which are based on how the enterprise back end provides the messages to the MobileFirst Server.

Two common ways exist to create an IBM MobileFirst Platform Foundation for iOS push notification architecture:

- The Java Message Service (JMS) polling method, in which messages are pulled from the JMS message queue and sent by the MobileFirst Server
- The enterprise back end method, in which an enterprise back end uses a MobileFirst adapter to deliver messages to a MobileFirst Server cluster

JMS polling architecture

This architecture relies on the enterprise backend to deliver messages to a single instance of MobileFirst Server by using a JMS message queue. The developer must create an IBM MobileFirst Platform Foundation for iOS JMS adapter, which pulls messages from the queue and calls the IBM MobileFirst Platform Foundation for iOS server-side push notification API to process the messages.

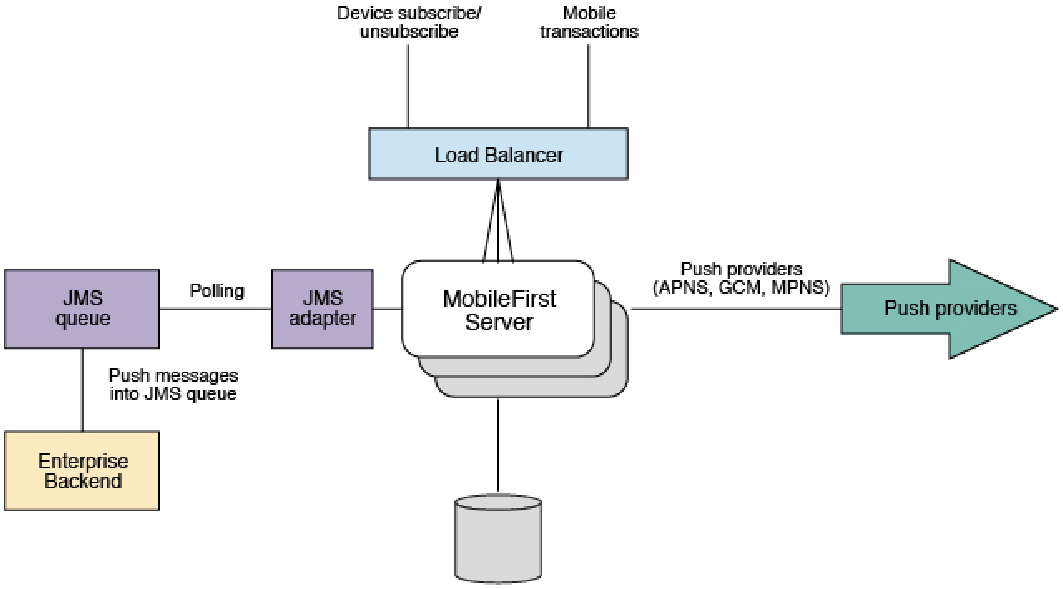


Figure 8-29. JMS polling push notification architecture

When this architecture is used, the flow is as follows:

1. Messages are put into the JMS queue by the enterprise backend.
2. The MobileFirst 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 MobileFirst Server instance pulls from the JMS queue and sends the push notifications. Even in a MobileFirst Server cluster, only one MobileFirst Server polls.

4. The process is implemented by using a MobileFirst JMS adapter, which functions as follows:
   - In a MobileFirst Server cluster, the single polling MobileFirst Server is selected randomly, by using the IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Server architecture

This architecture relies on the enterprise backend to deliver messages to a MobileFirst Server cluster by calling a MobileFirst adapter procedure.

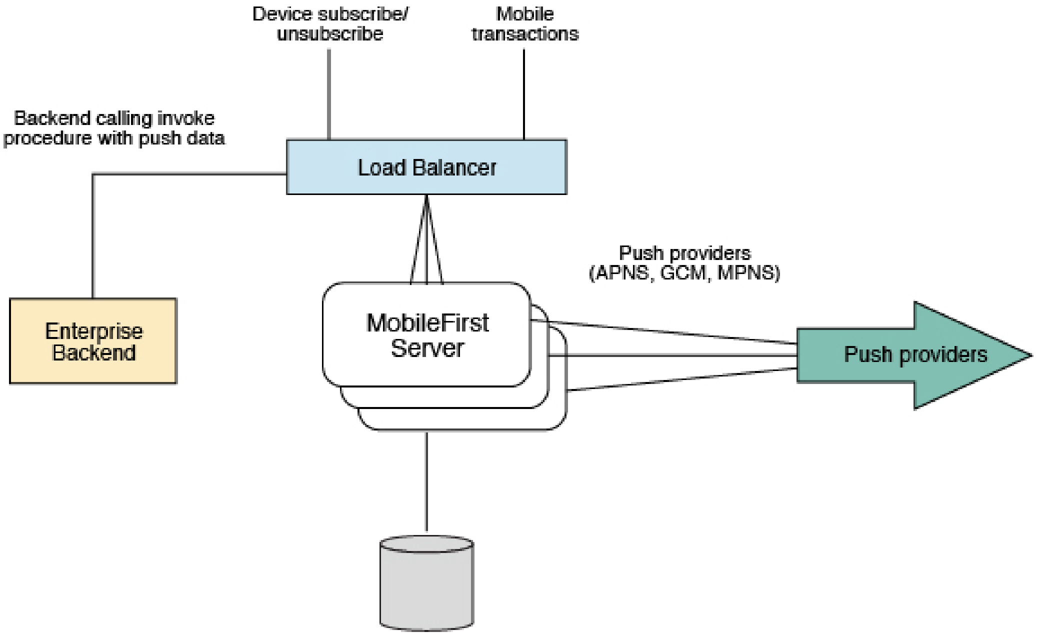

Device subscribe/ unsubscribe
Mobile transactions
Backend calling invoke procedure with push data
Load Balancer
Push providers (APNS, GCM, MPNS)
Enterprise Backend
MobileFirst Server
Push providers

Figure 8-30. Enterprise backend push notification architecture

When this architecture is used, the flow is as follows:

1. The request is routed to one of the MobileFirst Server instances, which sends a push message to a provider.
2. In this flow, all MobileFirst 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 MobileFirst 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 MobileFirst Server. You can mitigate this overhead by sending a number of messages together or by having the MobileFirst 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 MobileFirst Server.

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 MobileFirst 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. Choose either of the following:
   - For a Hybrid app, place the Apple APNS certificate file at the root of the application folder, along with the `application-descriptor.xml` file.
   - For a Cordova app, place the `.p12` file in the root of the Cordova project, along with the Cordova app `config.xml` file.
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.

Broadcast notifications

Broadcast notifications are notification messages that are targeted to all the devices that have the MobileFirst application installed and configured for push notifications.

Broadcast notifications are enabled by default with any MobileFirst application that is enabled for push notification. For more information about configuring your application for push notifications, see “Setting up push notifications” on page 8-172.

Any MobileFirst application that is enabled for push notification has a predefined subscription to the `Push.ALL` tag, which is used by MobileFirst Server to broadcast notification messages to all the devices. To disable broadcast notification for iOS native app, use `unsubscribeTag` method of WLPush class, with the tag name `Push.ALL`.

For more information about sending broadcast notification, see “Broadcast notification” on page 8-179 section in “Sending push notifications” on page 8-179.

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 MobileFirst 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 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 MobileFirst 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” on page 8-174.

For more information about sending event source-based notification, see Event source-based notification section in “Sending push notifications” on page 8-179.

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 MobileFirst 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 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 MobileFirst Server.

For iOS native application, use the methods of WLPush class.

Procedure

1. When the application first connects to the MobileFirst Server from the device, the device registers with a push service mediator and obtains a device token. This process is done automatically by IBM MobileFirst Platform Foundation for iOS.
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 MobileFirst Server that the device is permanently inaccessible.

Results

While the user subscription exists, the MobileFirst Server can produce push notifications for the subscribed user.

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 8-179.

Interactive notification

With interactive notification, when a notification arrives, users can take actions without opening the application. When an interactive notification arrives, the device shows action buttons along with the notification message. Currently, interactive notifications are supported on devices with iOS version 8 onwards. If an interactive notification is sent to an iOS device with version earlier than 8, the notification actions are not displayed.

Sending interactive push notification

Prepare the notification and send notification. For more information, see "Sending push notifications" on page 8-179.

You can set a string to indicate the category of notification with the notification object. Based on the category value, the notification action buttons are displayed.

To set the category in event source notifications, you have two options:

- Create a notification JSON object and set the category in that object:
  ```
  var notification = { badge:1, category: 'poll', ....};
  ```
- Create a notification object by using the WL.Server.createDefaultNotification API and set the category on the notification object:
  ```
  notification.APNS.category= 'poll';
  ```

For more information, see the WL.Server.createDefaultNotification and WL.Server.notifyAllDevices APIs in WL.Server class.

In broadcast, tag-based and Uni-cast notifications, set the type while you create the notification object:

```
notification.settings.apns.category = 'poll';
```

For more information, see the WL.Server.sendMessage API in WL.Server class.

Handling interactive push notifications in native iOS application

You must follow these steps to receive interactive notifications:

1. Enable the application capability to perform background tasks on receiving the remote notifications. This step is required if some of the actions are background-enabled.
2. In the AppDelegate (application: didRegisterForRemoteNotificationsWithDeviceTokenapplication:), set the categories before you set the deviceToken on WLPush Object.

```
if([application respondsToSelector:@selector(registerUserNotificationSettings:)]){
  UIUserNotificationType userNotificationTypes = UIUserNotificationTypeNone | UIUserNotificationTypeSound | UIUserNotifi

  UIMutableUserNotificationAction *acceptAction = [[UIMutableUserNotificationAction alloc] init];
  acceptAction.identifier = @"OK";
  acceptAction.title = @"OK";

  UIMutableUserNotificationAction *rejetAction = [[UIMutableUserNotificationAction alloc] init];
  rejetAction.identifier = @"NOK";
  rejetAction.title = @"NOK";

  UIMutableUserNotificationCategory *cateogory = [[UIMutableUserNotificationCategory alloc] init];
  cateogory.identifier = @"poll";
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContextDefault];
  [cateogory setActions:@[acceptAction,rejetAction] forContext:UIUserNotificationActionContextMinimal];
```

```
  NSSet *catgories = [NSSet setWithObject:cateogory];
  [application registerUserNotificationSettings:[UIUserNotificationSettings settingsForTypes:userNotificationTypes categori
}
```

3. Implement new callback method on AppDelegate:

```
-(void)application:(UIApplication *)application handleActionWithIdentifier:(NSString *)identifier forRemoteNotification:(NS
```

   This new callback method is invoked when the user clicks the action button.

4. Implement this method mustso that it perform the action that is associated with the specified identifier and executes the block in the **completionHandler** parameter.

Silent notifications

Silent notifications are notifications that do not display alerts or otherwise disturb the user. When a silent notification arrives, the application handing code runs in background without bringing the application to foreground. Currently, the silent notifications are supported on iOS devices with version 7 onwards. If the silent notification is sent to iOS devices with version lesser than 7, the notification is ignored if the application is running in background. If the application is running in the foreground, then the notification callback method is invoked.

Sending silent push notification

Prepare the notification and send notification. For more information, see "Sending push notifications" on page 8-179.

The three types of notifications that are supported for iOS are represented by constants DEFAULT, SILENT, and MIXED. When the type is not explicitly specified, the DEFAULT type is assumed.

For MIXED type notifications, a message is displayed on the device while, in the background, the app awakens and processes a silent notification. The callback method for MIXED type notifications gets called twice - once when the silent notification reaches the device and once when the application is opened by tapping on the notification.

To set the type in event source notifications, create notification object by using the WL.Server.createDefaultNotification API and set type on the notification object:

```
notification.APNS.type = "DEFAULT" | "SILENT" | "MIXED";
```

For more information, see the WL.Server.createDefaultNotification and WL.Server.notifyAllDevices APIs in WL.Server class.

If the notification is event source-based, the silent notifications are ignored if they arrive before the application registers the callback.

In Broadcast, Tag-based and Uni-cast notifications set the type while you create the notification object:

```
notification.APNS.type = "DEFAULT" | "SILENT" | "MIXED";
```

For more information, see the WL.Server.sendMessage API in WL.Server class.

If the notification is silent, the alert, sound, and badge are ignored.

Handling silent push notifications in native iOS application

You must follow these steps to receive silent notifications:

1. Enable the application capability to perform background tasks on receiving the remote notifications.
2. Implement new callback method on AppDelegate (application: `didReceiveRemoteNotification:fetchCompletionHandler:`) to receive silent notifications when the application is running on background.
3. In the callback, check whether the notification is silent or not by checking that the key content-available is set to 1.
4. After you finish processing the notification, you must call the block in the **`handler`** parameter immediately. Otherwise, your app will be terminated. Your app has up to 30 seconds to process the notification and call the specified completion handler block.

Tag-based notifications

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 8-180 section in “Sending push notifications” on page 8-179.

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.

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

Subscriptions tie together a device registration and a tag. When a device is unregistered from a tag, all associated subscriptions are automatically unsubscribed from the device itself. In a scenario where there are multiple users of a device, subscriptions should be implemented in mobile applications based on user login criteria. For example, the subscribe call is made after a user successfully logs in to an application and the unsubscribe call is made explicitly as part of the logout action handling.

To receive notifications that are targeted to a particular tag, you subscribe the application to the tags that you defined. For iOS native application, use the methods of WLPush class.

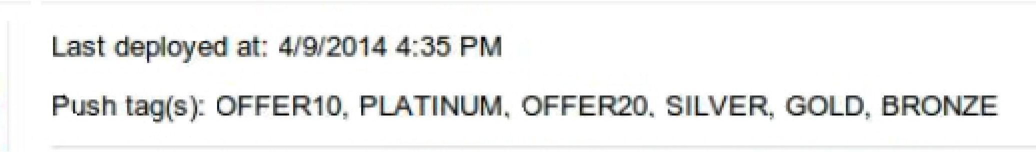
Procedure

1. When the application first connects to MobileFirst Server from the device, the device registers with a push service mediator and obtains a device token. This process is done automatically by IBM MobileFirst Platform Foundation for iOS.
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. Note that only successful authentications with a proper user ID will be associated.
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 MobileFirst Server that the device is permanently unaccessible.
5. For iOS native app, the notification comes to didReceiveRemoteNotification method that is defined by default in your `appDelegate.m` file.

Results

While the tag subscription exists, MobileFirst 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 MobileFirst Operations 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 8-180 section in "Sending push notifications" on page 8-179.

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 MobileFirst application is enabled for push notifications. For more information about configuring your application for push notifications, see "Setting up push notifications" on page 8-172.

For more information about sending unicast notification, see "Unicast notification" on page 8-180 section in "Sending push notifications."

Note: For a Unicast notification, the payload object does not contain any tag.

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 8-173.

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 8-174.

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) {
  var userSubscription = WL.Server.getUserNotificationSubscription('PushAdapter.PushEventSource',
  if(userSubscription === null) {
    return{ result: "No subscription found for user :: "+ userId };
  }

  var badgeDigit = 1;
  var notification = WL.Server.createDefaultNotification(notificationText, badgeDigit, {custom:"da
    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 8-177.

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.

The **`userId(s)`** must be the user IDs that were used to subscribe to the push notification event source. The user ID in the user subscription can come from the underlying security context or be a user ID explicitly set by your mobile app. A user ID explicitly set by your mobile app is also called an application user ID (**`appUserId`**).

Note: For a Unicast notification, the payload object does not contain any tag.

For more information about implementing application user IDs in your mobile app, see WLClient class.

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, A (Apple), in the **`notificationOptions.target.platforms`** object.

Restriction

Restriction: The `sendMessage` method does not support SMS notification. For more information, see "Sending SMS push notifications" on page 8-181.

REST Services APIs

You can use REST Services APIs to work with Push notifications.

Use REST API Administration Services for adapters and applications.

Use REST API Runtime Services for applications deployed outside of the MobileFirst Server.

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 devices that support SMS functions. IBM MobileFirst Platform Foundation for iOS includes the capability to send SMS notifications to all platforms that provide SMS support.

Procedure

1. An SMS notification infrastructure is set up.

   A MobileFirst 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 MobileFirst adapter, by using the client-side `WL.Client.Push.subscribeSMS` method.
3. The user subscription to the event source is registered at the MobileFirst Server.
4. When the back-end service must notify the user, it calls a method in the MobileFirst 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 MobileFirst Operations 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 MobileFirst Platform Foundation for iOS.

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.

Web-based SMS subscription

Subscription, and unsubscription, to SMS notifications can be performed by making HTTP GET requests to the subscribe SMS servlet. The subscribe SMS servlet can be used for SMS subscriptions without the requirement for a user to have an app installed on their device.

Enter the following URL to access the subscribe SMS servlet:

```
http://<hostname>:<port>/<context>/subscribeSMS
```

This URL can be used to subscribe and unsubscribe.

You must create an application and an event source within an adapter and deploy them on the IBM MobileFirst Platform 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 8-16. 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.

Note: If any parameter value contains special characters, this parameter must be encoded by using URL encoding, also known as percent encoding, before the URL is constructed. Parameter values containing only the following characters do not need to be encoded:

a-z, A-Z, 0-9, period (.), plus sign (+), minus sign (-), and underscore (_)

Subscriptions that are created by using the subscribe SMS servlet are independent of subscriptions that are created by using a device. For example, it is possible to have two subscriptions for the same phone number and user name; one created by using the device and one created by using the subscribe SMS servlet. If there are two subscriptions for the same phone number and user name, unsubscription by using the subscribe SMS servlet unsubscribes only the subscription that is made through the subscribe SMS servlet. However, unsubscription by using the IBM MobileFirst Platform Operations 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 MobileFirst Platform Foundation for iOS 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, MobileFirst 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>
```

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 MobileFirst client framework can subscribe to.

About this task

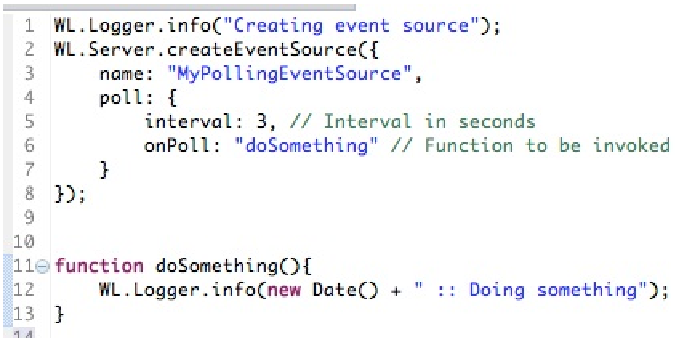
The MobileFirst adapter framework provides the ability to implement event sources, which can be used to generate notification events such as push notifications. However, notifications must be retrieved from a back-end system before they can be sent out. Event sources can either poll notifications from the back-end system, or wait for the back-end system to explicitly push a new notification.

This task describes how to create a polling event source, and use it to send push notifications. A polling event source is a long-running task that has the following mandatory properties:

- Event source name
- Polling interval
- Polling function

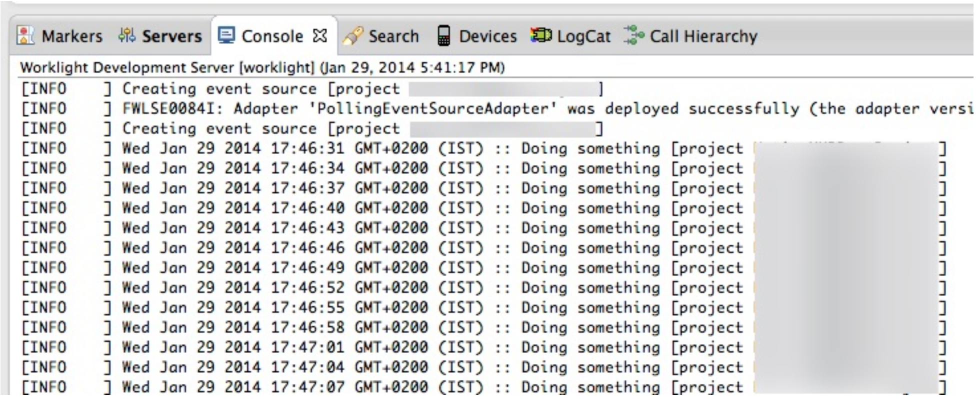
Procedure

- Consider the following simple example. The diagram shows a sample for a basic polling event source:



```
WL.Logger.info("Creating event source");
WL.Server.createEventSource({
    name: "MyPollingEventSource",
    poll: {
        interval: 3, // Interval in seconds
        onPoll: "doSomething" // Function to be invoked
    }
});


function doSomething(){
    WL.Logger.info(new Date() + " :: Doing something");
}
```

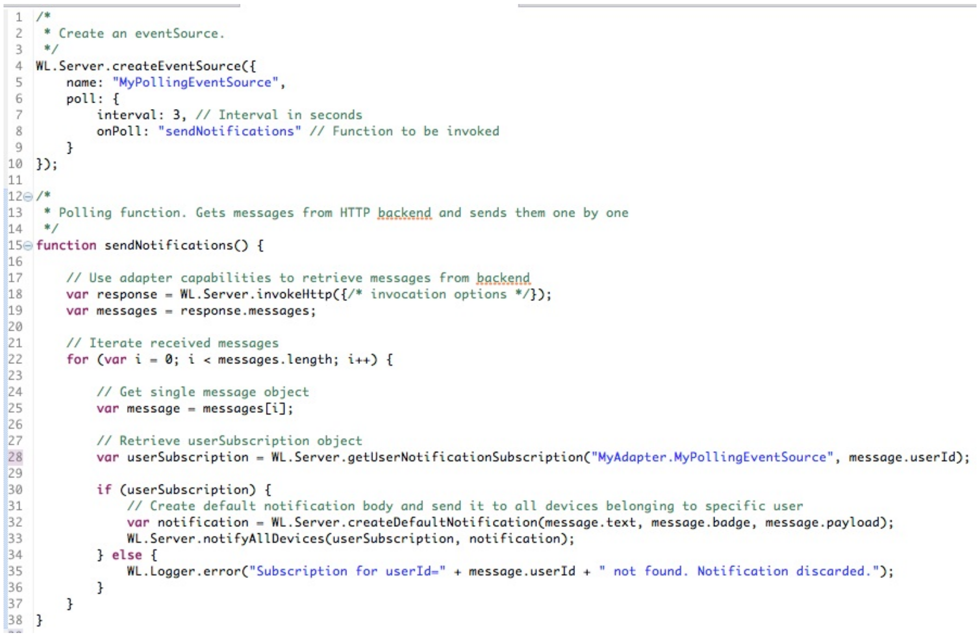
  The doSomething() function is invoked every three seconds. If you deploy this adapter to the MobileFirst Server, you see the following logs in the server console:



```
Worklight Development Server [worklight] (Jan 29, 2014 5:41:17 PM)
[INFO    ] Creating event source [project [redacted]]
[INFO    ] FWLSE0084I: Adapter 'PollingEventSourceAdapter' was deployed successfully (the adapter versi
[INFO    ] Creating event source [project [redacted]]
[INFO    ] Wed Jan 29 2014 17:46:31 GMT+0200 (IST) :: Doing something [project [redacted]]
[INFO    ] Wed Jan 29 2014 17:46:34 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:37 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:40 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:43 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:46 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:49 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:52 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:55 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:46:58 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:47:01 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:47:04 GMT+0200 (IST) :: Doing something [project ]
[INFO    ] Wed Jan 29 2014 17:47:07 GMT+0200 (IST) :: Doing something [project ]
```

  The log shows that the doSomething() function is invoked at 3-second intervals.

- This second example shows a more realistic example of a polling event source:



```
1  /*
2   * Create an eventSource.
3   */
4  WL.Server.createEventSource({
5      name: "MyPollingEventSource",
6      poll: {
7          interval: 3, // Interval in seconds
8          onPoll: "sendNotifications" // Function to be invoked
9      }
10 });
11
12 /*
13  * Polling function. Gets messages from HTTP backend and sends them one by one
14  */
15 function sendNotifications() {
16
17     // Use adapter capabilities to retrieve messages from backend
18     var response = WL.Server.invokeHttp({/* invocation options */});
19     var messages = response.messages;
20
21     // Iterate received messages
22     for (var i = 0; i < messages.length; i++) {
23
24         // Get single message object
25         var message = messages[i];
26
27         // Retrieve userSubscription object
28         var userSubscription = WL.Server.getUserNotificationSubscription("MyAdapter.MyPollingEventSource", message.userId);
29
30         if (userSubscription) {
31             // Create default notification body and send it to all devices belonging to specific user
32             var notification = WL.Server.createDefaultNotification(message.text, message.badge, message.payload);
33             WL.Server.notifyAllDevices(userSubscription, notification);
34         } else {
35             WL.Logger.error("Subscription for userId=" + message.userId + " not found. Notification discarded.");
36         }
37     }
38 }
```

The sample includes the following key elements:

- Lines 7 - 8: The polling event source continuously invokes a `sendNotifications()` function with a 3-second interval.
- Lines 18 - 19: Every time the `sendNotifications()` function is invoked it requests messages data from the back-end. The sample shows an HTTP back-end, but it could be any other type of back-end that MobileFirst adapters support; for example, SQL. The code assumes that the following JSON markup is returned by the back-end. However, since the MobileFirst 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 MobileFirst 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 MobileFirst Server, over an SMS channel. SMS messages that originate from the mobile device can be sent to the MobileFirst Server through an external SMS gateway. The MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst notification framework. For more information, see Push notification.

The two-way SMS architecture is summarized in the following figure:

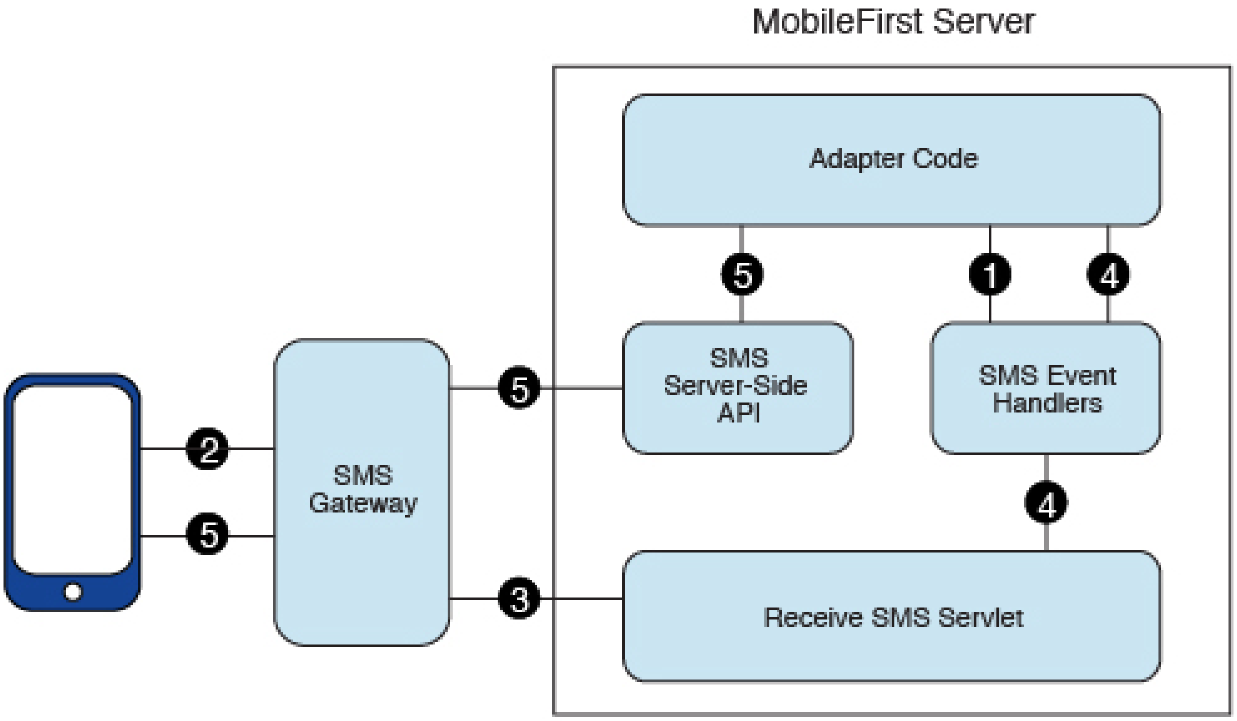


Figure 8-31. Two-way SMS architecture

1. The adapter registers SMS event handlers on the MobileFirst Server.
2. SMS messages are sent from mobile devices to the SMS gateway, which is configured with an SMS servlet of MobileFirst Server.
3. The SMS gateway forwards SMS messages to a configured MobileFirst URL.
4. An SMS servlet on MobileFirst 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.

Troubleshooting push notification problems

To resolve iOS push notification issues, download the JCE policy file and update it for the server environment.

About this task

The push notification fails to send, and you see the following exception in the server log:

```
com.notnoop.exceptions.InvalidSSLConfig: java.io.IOException: Error in loading the keystore: Private

at com.notnoop.apns.internal.Utilities.newSSLContext(Utilities.java:88)

at com.ibm.pushworks.server.notification.apns.ApplicationConnection.createBuilderWithCertificate(Appl
at com.ibm.pushworks.server.notification.apns.ApplicationConnection.<init>(ApplicationConnection.java

...
```

Procedure

1. Download the unrestricted version of the JCE policy files.
   a. Log in to Unrestricted SDK JCE policy files.
   b. Select **Unrestricted JCE Policy files for SDK** for all later versions (Version 1.4.2 and later).
   c. Click **Continue** and finish the download process.

      The .zip archive contains 3 files:
      - readme.txt
      - local_policy.jar
      - US_export_policy.jar
2. Update the JCE policy files for the server environment.
   a. Stop the server.
   b. Use the new local_policy.jar file and the new US_export_policy.jar file to replace the old local_policy.jar file and the US_export_policy.jar file that are in the *jdk_path*/jre/lib/security folder.

      Note: The *jdk_path* might be bundled with the server.
   c. Restart the server.

MobileFirst security framework

A collection of topics that describe OAuth-based and classic security features in IBM MobileFirst Platform Foundation for iOS.

OAuth-based security model

Topics that describe OAuth-based security features in IBM MobileFirst Platform Foundation for iOS.

Overview

The OAuth 2.0 protocol is based on acquiring an access token, which encapsulates the authorization that is granted to the client. In that context, the IBM MobileFirst Platform Server serves as an authorization server and is able to generate such tokens. The client can then use these tokens to access resources on a resource server, which can be either the MobileFirst Server itself, or an external server. The resource servers perform validation on the token to make sure that the client can be granted access to the requested resource. This separation between resource server and authorization server in the new OAuth-based model allows you to enforce IBM MobileFirst Platform Foundation for iOS security on resources that are running outside the MobileFirst Server.

To support compatibility with earlier versions, the classic (pre-V7.0) security model is still in use in the flows that are based on the existing APIs. The new client APIs trigger flows that conform to the OAuth-based security model. V7.0 provides seamless integration between the two security models. The platform allows you to mix classic and new APIs in the same application, while keeping a consistent security context on the server side.

End-to-end authorization flow

The new end-to-end authorization flow is based on the OAuth specification and comprises the following phases:

1. Acquiring a token (see Figure 1): In this phase, the client obtains a token from the authorization server, following these steps:
   a. Registration: This phase occurs once in the lifetime of a mobile app that is installed on a device. In this phase, the client registers itself with the MobileFirst Server. When application authenticity has been configured, it is activated during registration.
   b. Authorization: In this phase, the client has to undergo specific security checks, according to the scope of the authorization request. These checks are implemented using the building blocks of the classic security model: authentication realms are used on the server side, and challenge handlers are used on the client side. All the security checks supported by IBM MobileFirst Platform Foundation for iOS can be used in this phase (built-in realms such as remoteDisable and others, custom realms, and adapter-based authentication).
   c. Token generation: After successful authorization, the client is redirected to the token endpoint, where it is authenticated using the PKI trust that was established during the registration phase. The endpoint then generates two sets of tokens and sends them back to the client: an access token, which encapsulates all the security checks that the client has passed in the authorization phase and an ID token, which contains information regarding the user and device identity of the client.

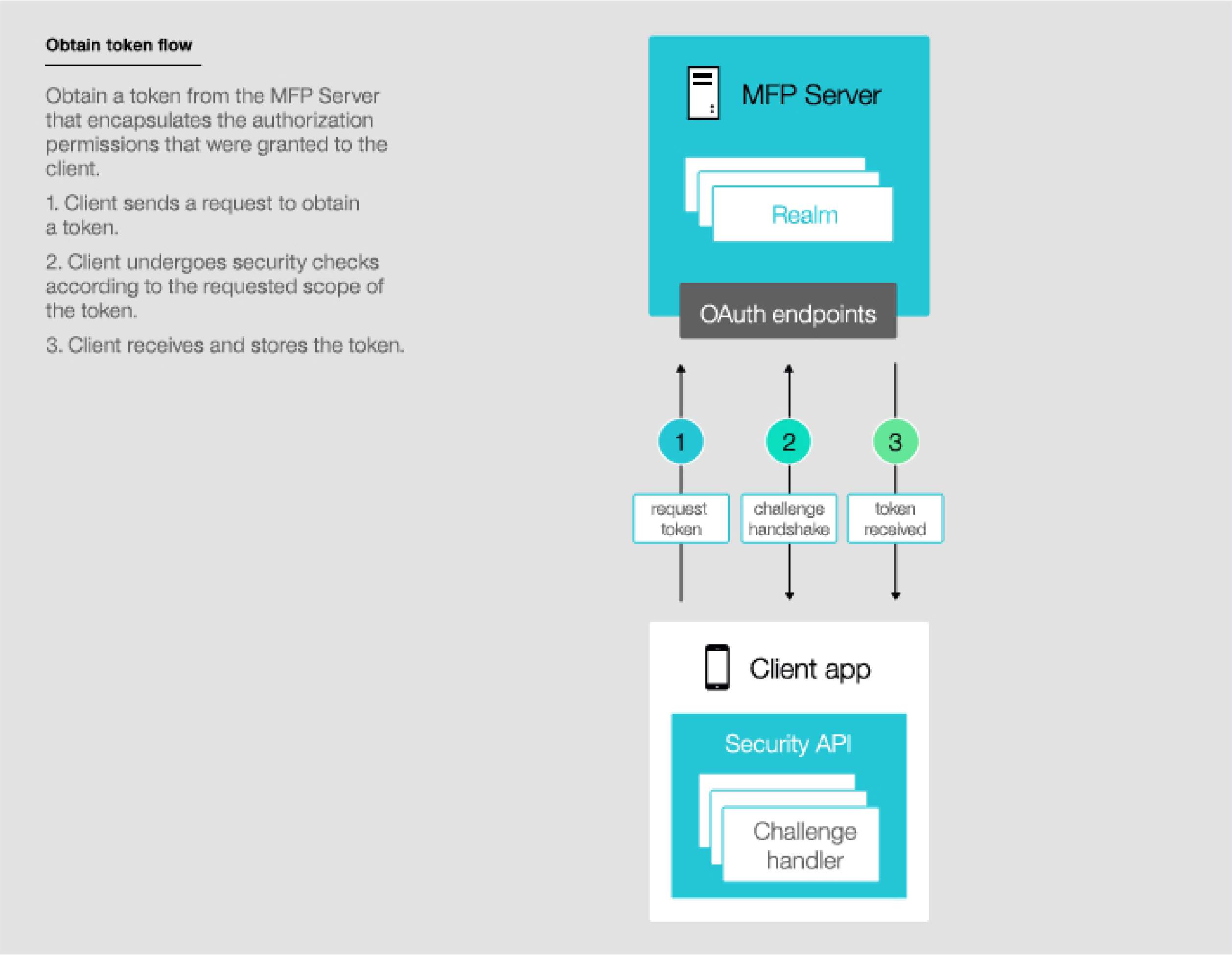
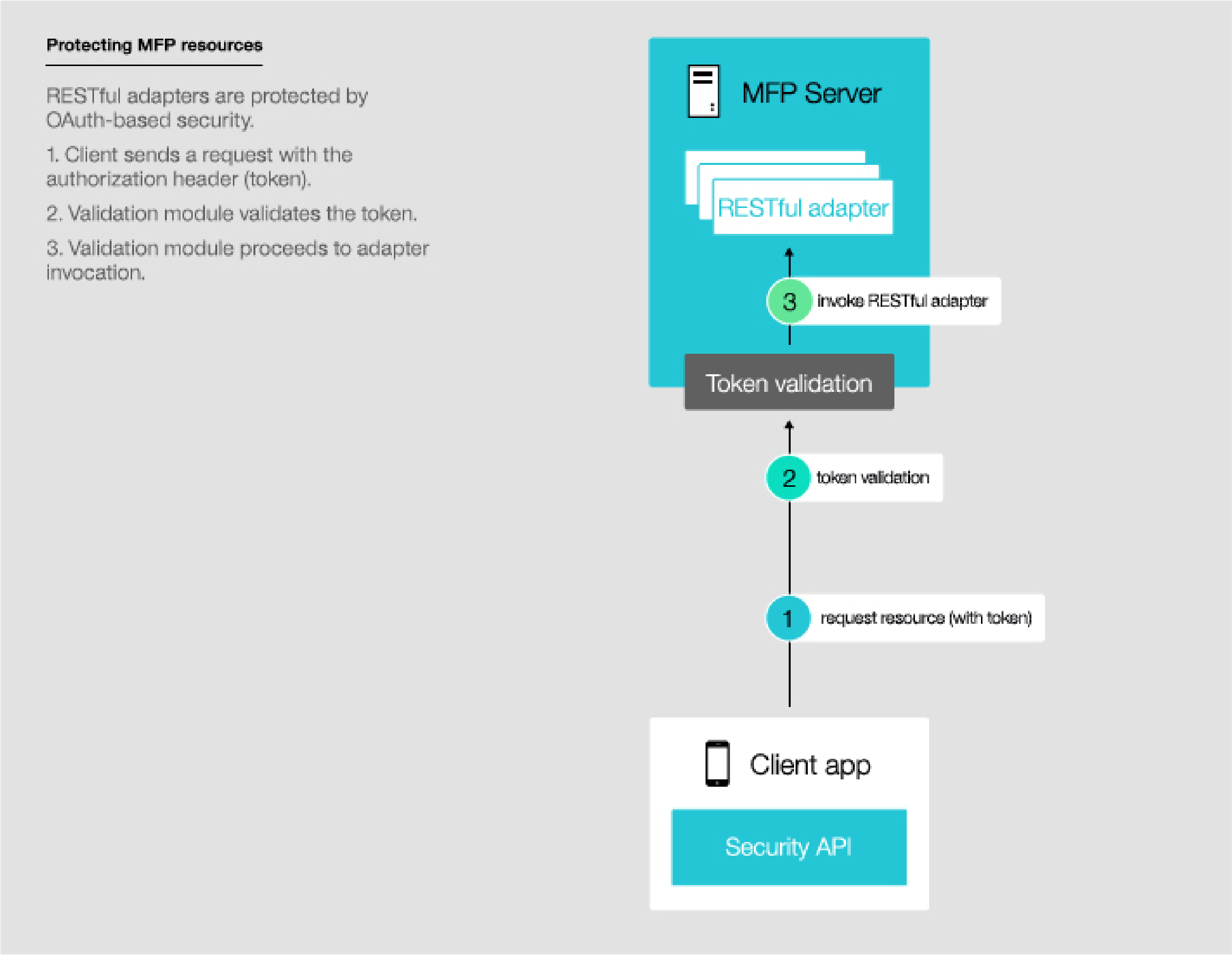

Obtain token flow
Obtain a token from the MFP Server that encapsulates the authorization permissions that were granted to the client.
1. Client sends a request to obtain a token.
2. Client undergoes security checks according to the requested scope of the token.
3. Client receives and stores the token.
MFP Server
Realm
OAuth endpoints
1
2
3
request token
challenge handshake
token received
Client app
Security API
Challenge handler

Figure 8-32. Obtain token flow

2. Using a token to access protected resources: Protected resources can run on the MobileFirst Server (see Figure 2) or on external servers (see Figure 3). Resources on external servers can be protected by using the validation modules that are provided with IBM MobileFirst Platform Foundation for iOS: the Node.js validation module for protecting Node.js resources and the Trust association interceptor (TAI) for protecting Java resources, or by developing your own custom filter as shown in the Token validation endpoint example. These modules validate the access token and ID token in incoming requests, making sure that only authorized clients are served. To consume such protected resources, the client has to first acquire the token from the MobileFirst Server, and then add the tokens as an authorization header to outgoing requests.

   The new client APIs enable the client to access protected resources while transparently handling the handshake with the authorization server.



Protecting MFP resources

RESTful adapters are protected by OAuth-based security.

1. Client sends a request with the authorization header (token).

2. Validation module validates the token.

3. Validation module proceeds to adapter invocation.

Figure 8-33. Protecting a resource on the MobileFirst Server

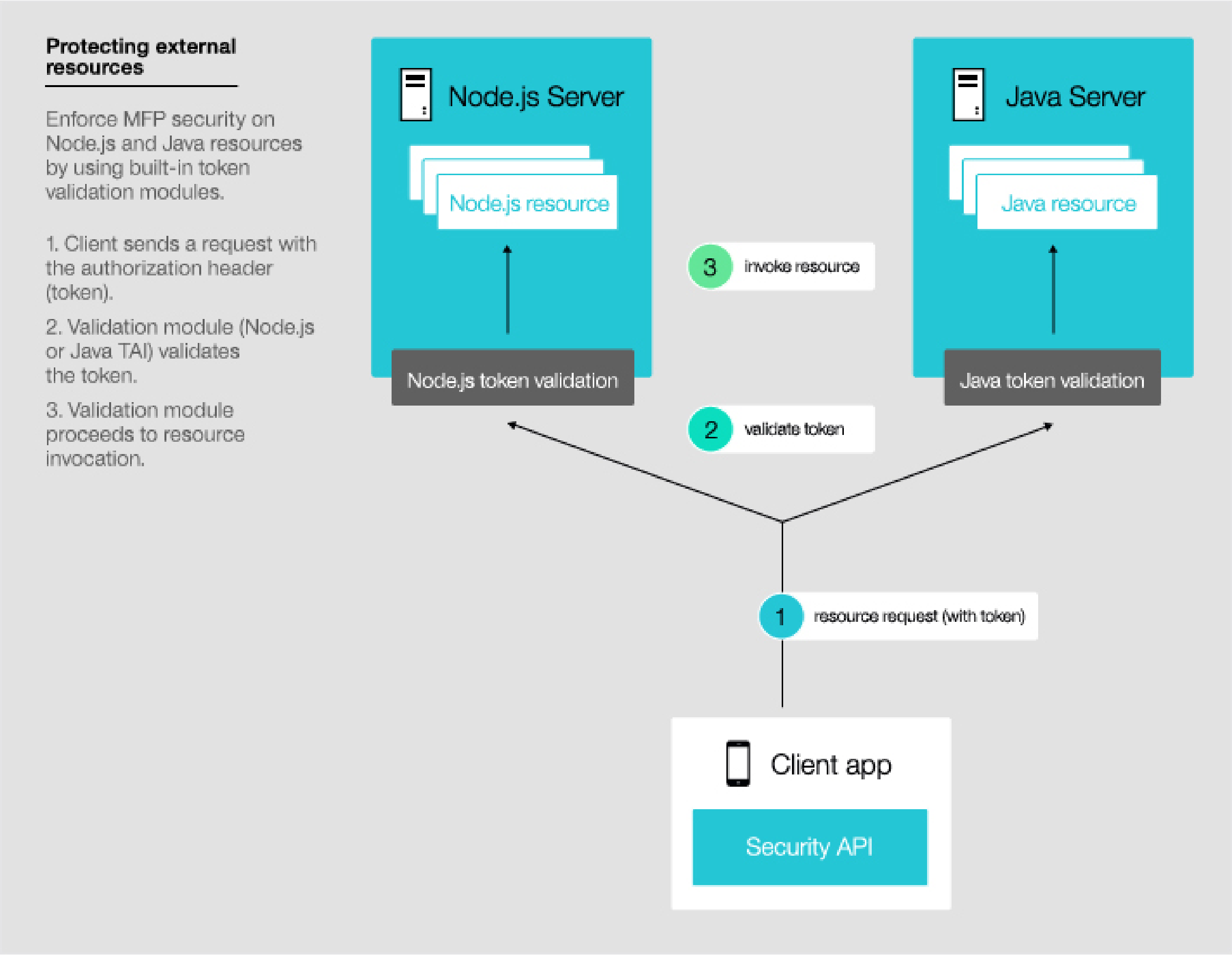

Protecting external resources
Enforce MFP security on Node.js and Java resources by using built-in token validation modules.
1. Client sends a request with the authorization header (token).
2. Validation module (Node.js or Java TAI) validates the token.
3. Validation module proceeds to resource invocation.
Node.js Server
Node.js resource
Node.js token validation
Java Server
Java resource
Java token validation
3 invoke resource
2 validate token
1 resource request (with token)
Client app
Security API

Figure 8-34. Protecting a resource on an external server

Protecting resources with OAuth-based security

It is possible to enforce IBM MobileFirst Platform Foundation for iOS security on resources running on the MobileFirst Server, as well as on resources running on any, external resource server. IBM MobileFirst Platform Foundation for iOS V7.0 provides several types of validation modules to enforce security on resources. These modules perform the following validation logic:

- Integrity: Digital signature for both ID token and access token is verified.
- Token expiration: Both tokens are tested for expiration. An expired token will fail the validation.
- Scope: Required scope for the resource is verified. The token has to contain the scope tokens (realm names) that are defined in the scope, and these scope tokens have to be valid (meaning, not expired).
- Mandatory scope: In addition to the scope defined by the resource, the access token must contain the valid scope that is required by the application. This scope contains the realms, which are defined in the security test that is protecting the application.

Note that each scope token (realm name) has its own expiration time, which can be different than the expiration time of the containing access token. To pass validation, both a scope token and its containing token must not be expired. The expiration period of a scope token (realm name) is defined by the expiration attribute of the login module that is associated with the realm. For more information about configuring expiration of login modules, see “Configuring login modules” on page 8-263. For information about configuring access token expiration, see Application descriptor of native API applications.

Consider the following example: a resource /somePath/myResource/doSomething is protected by the scope {myCustomRealm myLoginRealm}. All requests to this resource will be intercepted by the validation module, so a valid token with a scope that contains myCustomRealm and myLoginRealm must be present in the request. In case such a token exists, the request is granted, and the resource is consumed. Otherwise, the validation module rejects the request. In this case, the client (through the client API) interacts with the MobileFirst Server to obtain the token, which requires successful authentication to the specified realms. After acquiring the token, the client resends the request, which now passes the token validation, and the resource is consumed.

Protecting internal resources - RESTful adapters

IBM MobileFirst Platform Foundation for iOS V7.0 enables you to enforce security on resources that are running on the MobileFirst Server, that is, Java and JavaScript RESTful adapters. For more information, see Security configuration of a JAX-RS resource.

Protecting external resources

With IBM MobileFirst Platform Foundation for iOS V7.0, you can enforce MobileFirst security on resources running outside the MobileFirst Server. To this end, you can use the built-in Node.js and Java validation modules, or implement your own custom validation module in the technology of your choice, using the online validation endpoint:

Node filter

The passport-mfp-token-validation module provides a passport validation strategy for protecting apps on a Node.js server. For more information, see “Protecting resources on Node.js servers” on page 8-197.

TAI filter

You can use the IBM MobileFirst Platform Foundation for iOS OAuth trust association interceptor (TAI) to protect application resources on WebSphere Application Server or WebSphere Application Server Liberty. For more information, see “Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty” on page 8-199.

Custom filter

The token validation endpoint on the MobileFirst Server validates tokens that are issued by the authorization server. For more information, see “Protecting resources with the token validation endpoint” on page 8-205.

Specifying the user identity realm in OAuth-based flows

Although user authorization may include authorization with several realms, only one of these realms is defined as the user identity realm, and that realm determines the user identity. In an OAuth-based flow, the user identity realm is set according to the userIdentityRealms definition in the Application descriptor of native API applications. Note that in the classic (pre-V7.0) flows, the user identity realm is selected according to the definition in the security test. See customSecurityTest. In OAuth-based flows, information about the user identity, which is set by the userIdentityRealms attribute, is part of the data contained in the "ID token" on page 8-196.

Combining classic and OAuth-based security models

MobileFirst Server V7.0 provides a smooth integration between the classic and OAuth-based security models.

Essentially, both flows share the same security context on the server side, and so any security state of a specific realm (such as expiration, or login status) is consistently shared between the flows. This allows you, the developer, to mix classic and OAuth-based security APIs, while providing a unified experience for the end user.

Consider, for example, a client application that calls an adapter procedure protected by some realm myCustomRealm and then retrieves a token for a scope that includes the same realm. In this case, the user will have to pass the myCustomRealm security check only on the first call, and will get the relevant token on the second call, without having to re-authenticate.

Some issues need to be considered when using both models in the same flow, or on the same resource:

- Device SSO:

  The configuration of the Device SSO feature is based on the device and user identity realms that are defined in the security test, which is protecting the resource. However, OAuth resources are not protected by security tests and do not have a single defined user realm per resource, so the standard MobileFirst device SSO behavior does not apply to them. For more information, see "Device single sign-on (SSO)" on page 8-291..
- The login/logout API:

  The WLClientWLClient login/logout API enables a user to log in to and log out of a specific realm, by updating the server side security state. However, in the new OAuth-based security model, security credentials are also kept in the access token on the client side. The result is that using this API will cause an inconsistent state, for example, in which the client is logged out of a realm on the server side but still holds a valid token for that realm on the client side. To solve this inconsistency, it is recommended to re-obtain the access token, by using the obtainAuthorizationHeaderForScope method, after successful login or logout.

  For example, consider a client that passed the security checks for Realm1 and Realm2, and later calls logout(Realm2). In this case, the access token on the client would still contain the security credentials for both Realm1 and Realm2, and the client could use this token to access protected resources. To refresh the token, that is, to obtain a token for Realm1 only, the client calls obtainAuthorizationHeaderForScope without the logged out realm Realm2.

  iOS example:

```
[[WLClient sharedInstance]logout:@"MyRealm" withDelegate:self];

-(void)onSuccess:(WLResponse *)response{
    // re-obtain the token
    [[WLAuthorizationManager sharedInstance]obtainAuthorizationHeaderForScope:nil completionHandler:^(WLResponse *response, NSError *error) {
      // successful logout logic
    }];
}
```

Test token endpoint

The test token endpoint enables you to get a valid token via a REST call, without having to set up a mobile client. For more information, see "The test token endpoint" on page 8-208.

Public key endpoint

The public key endpoint gets the server's public key. The public key is used, for example, for the verification of digitally signed OAuth tokens. The path for the endpoint is as follows:

```
/authorization/v1/publickey
```

The endpoint returns the public key information using the JWK standard format. The following is an example of a result returned by this endpoint:

```
{
    "e": "AQAB",
    "n": "AM0Dd7xAdv6H-ygL7r8qCLdE-3I2kk45zgZtDd_qs8fvnYfdiqTSV4_2t6OGG8CV5Ce41PMpIwmL410X9IZnvhxoYiFcMSaOeIqoe-rJA0uZuw
    "kty": "RSA"
}
```

Note: The server uses an RSA key type, and therefore the `kty` attribute is always RSA.

Preview mode

Note: Preview mode is not available in production environments.

Exposed endpoints

You can enable white- and blacklists to the endpoints of the MobileFirst Server. For a list of endpoints for use with the OAuth-based security model, see "Endpoints of the MobileFirst Server production server" on page 6-333.

Security client APIs

IBM MobileFirst Platform Foundation for iOS V7.0 provides client APIs that conform to the new OAuth-based security model.

- There is an API for consuming protected resources, which transparently supports the interaction with the MobileFirst Server (obtaining the token requested by the resource).
  - In Objective-C: WLResourceRequest
- In addition, APIs are provided to create a custom client implementation that is able to interact with the validation modules and MobileFirst Server. See the following example:
  - "Custom requests to resources using Objective-C" on page 8-211

OAuth-based tokens

Learn about token response, MobileFirst Server authorization headers, access tokens, and ID tokens.

Access token

The access token is a digitally signed token that describes the authorization permissions of a client. The token contains the following noteworthy attributes:

- expiration: Expiration of the access token in seconds. An expired token is not valid.
- scope: Scope of the authorization permissions of this token. The scope comprises a list of scope tokens.
- scope tokens: Scope tokens are mapped to MobileFirst realms (a scope token is the name of a MobileFirst realm). Each such scope token represents a successful authentication that the client has undergone during the authorization phase. Each scope token has an expiration of its own (in seconds).

ID token

The ID token is a digitally signed token and implements the OIDC specification. It contains information about the device, application, and user identity. The device and application identity are defined in the registration phase and are always present in the ID token. The user identity is defined according to userIdentityRealms, as described in Application descriptor of native API applications. In case there is no such realm defined, the ID token contains no user identity, meaning that the user is “anonymous”.

To learn more about using the data in the ID token, see Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty, “Protecting resources on Node.js servers” on page 8-197,“Protecting resources with the token validation endpoint” on page 8-205, and “Implementing the JAX-RS service of the adapter” on page 8-27.

MobileFirst Server authorization header

After the client has acquired an access token, it can use the token in requests to protected resources. As described in the OAuth 2.0 Bearer Token Usage specification, the client uses the "Bearer" authentication scheme to transmit the access token in the authorization HTTP header. The following is an example of the value of the authorization header:

```
Authorization: Bearer yI6ICJodHRwOi8vc2VydmVyLmV4YW1...
```

Successful token response

The access token and ID token are generated in the token endpoint after the authorization phase and sent to the client.

A successful token response from the MobileFirst authorization server looks like this:

```
HTTP/1.1 200 OK
    Content-Type: application/json
    Cache-Control: no-store
    Pragma: no-cache
{
    "token_type": "Bearer",
    "expires_in": 3600,
    "access_token": "yI6ICJodHRwOi8vc2VydmVyLmV4YW1...",
    "id_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6Ij..."
    "scope": "wl_directUpdateRealm wl_remoteDisableRealm wl_antiXSRFRealm wl_deviceNoProvisioningRealm wl_anonymousUserRealm"
}
```

Note:

- token_type is always "Bearer", as defined in the OAuth 2.0 Bearer Token Usage specification.
- expires_in is the expiration time of the access token, in seconds.

- scope is the list of the scope tokens (realms) that is included in the access token.

Protecting external resources

You can protect application resources that are located on WebSphere Application Server, WebSphere Application Server Liberty or Node.js servers by using OAuth security.

Protecting resources on Node.js servers:

You can protect your resources that are running on Node.js servers with OAuth-based IBM MobileFirst Platform Foundation for iOS security.

mfpStrategy

The passport-mfp-token-validation npm module provides a passport validation strategy and a verification function to validate access tokens and ID tokens that are issued by the MobileFirst Server .

passport.use (new mfpStrategy(options));

The **options** parameter contains one or more of the following options:

- **publicKeyServerUrl**: (Mandatory) Specifies the URL of the MobileFirst Server from which the public key is retrieved to verify the tokens.

  Note: Alternatively, you can pass the public key server URL as a parameter to the passport.authenticate method. This method is used in the Example.)
- **scope**: Space-separated string to define the list of realm names that are required for accessing the resource. If no scope is specified, only the mandatory scope is checked in the token.

  Note: Alternatively, you can pass the scope as a parameter to the passport.authenticate method. (See Example.)
- **cacheSize**: The maximum number of tokens allowed. The default value is 500.
- **logger**: Defines a logger instance. The default value is the IBM default logger, which outputs log messages to the console.
- **analytics.onpremise**: Use this parameter to specify where to send the analytics logs. Logs are displayed in the MobileFirst Operational Analytics console.
  - **url**: The url that specifies the location of the operational analytics server. For example, http://localhost:10080/worklight-analytics-service/data.
  - **username**: The user name if credentials are required.
  - **password**: The password if credentials are required.

  For more information about operational analytics, see "Operational analytics" on page 13-6.

For more information about npm passports, see Passport Readme. For more information about the **passport.authenticate** method, see Authenticate.

Example

The following example shows how to use **mfpStrategy** in a node application:

```
var express = require('express'),
    passport = require('passport-mfp-token-validation').Passport,
    mfpStrategy = require('passport-mfp-token-validation').Strategy;
    //the configuration ('config') is optional if you wish to report
```

```
//events to the Analytics Server.
var config = {
  url : 'http://localhost:10080/worklight-analytics-service/data',
  username : 'admin',
  password : 'admin'
};

passport.use(new mfpStrategy({publicKeyServerUrl:'http://localhost:10080/WLProject',
analytics : {onpremise: config}}));

var app = express();
app.use(passport.initialize());

// protect API with mfpStrategy using scope Realm1 Realm2 Realm3
  app.get('/v1/apps/:appid/service', passport.authenticate('mobilefirst-strategy',
       {session: false , scope: 'Realm1 Realm2 Realm3' }),
    function(req, res){
           res.send(200, req.securityContext);
    }
 );

 app.listen(3000);
```

To start the example, issue the following commands:

```
$ npm install express
$ npm install passport
$ npm install passport-mfp-token-validation
```

Token verification

The passport-mfp-token-validation module verifies the authorization header of the request. The authorization header consists of the following elements:

`Bearer` *`Access_token ID_token`*

Where

`Bearer` (Mandatory) Is the required string for the token type, as defined in the OAuth 2.0 specification.

`Access_token`
(Mandatory) Encapsulates all of the security checks that the client has passed in the authorization phase.

`ID_token`
(Optional) Contains information about the user and device identity of the client.

`Bearer` and *`Access_token`* are mandatory. *`ID_token`* is optional. The passport-mfp-token-validation module verifies the token with the public key that is retrieved from the authorization server. If the token is verified successfully, the `securityContext` and `user` objects are attached to the request object.

securityContext
After a successful validation, a security context object is added to the current request.

The securityContext object contains the following fields:

- **`imf.sub`**: The **`sub`** value of the ID token or the unique ID of the client if there is no ID token.
- **`imf.user`**: The **`user`** value that is extracted from the ID token. If there is no ID token, this field holds a blank object.

- **imf.device**: The **device** value that is extracted from the ID token. If there is no ID token, this field holds a blank object.
- **imf.application**: The **application** value that is extracted from the ID token. If there is no ID token, this field holds a blank object.

user The user object in the request is returned by the passport framework. Its value is the same as the value of imf.user in the securityContext object.

Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty:

You can protect your resources that are running on WebSphere Application Server or WebSphere Application Server Liberty servers with OAuth-based IBM MobileFirst Platform Foundation for iOS security.

Overview

To enable MobileFirst OAuth security for resources on WebSphere Application Server or WebSphere Application Server Liberty servers, complete the following steps:

1. Configure OAuthTAI
2. Set the security role
3. Install and configure OAuthTAI on the server
4. Add the tokens to the authorization header
5. Use the security context to access resources

Configure OAuthTAI

You configure the OAuthTAI feature by using XML. For WebSphere Application Server Liberty, you add the XML to the server.xml file. For WebSphere Application Server, you save it in a separate file.

The OAuthTAI element is the root element. The following sections describe the OAuthTAI attributes and subelements:

OAuthTAI attributes

cacheSize

(Optional) Maximum number of tokens allowed. The default is 500.

id

(Mandatory) Defines a unique id of OAuthTAI.

OAuthTAI subelements

securityConstraint

(Mandatory) The value of securityConstraint must be a number 1 to *n*.

securityConstraint attributes

securedURLs

(Mandatory) If multiple URLs are specified in securedURLs, separate each URL by a space. The following types of URLs are supported:

- Exact match URL. For example,/context-root/a/xyz
- Path match URL. For example, /context-root/*

- Wild card match URL. For example,/context-root/*/xyz

httpMethods

(Optional) Default value: ALL. The httpMethods attribute is case insensitive. Its value can be one or more of the following strings: ALL, DELETE, GET, POST,PUT,HEAD,OPTIONS,TRACE, or CONNECT. If multiple methods are specified, separate the methods by a space.

scope

(Optional) If scope is not specified, the only checking that is done is to make sure that the mandatory scopes are not expired.

The following is an example:

```
<?xml version="1.0" encoding="UTF-8"?>
   <OAuthTAI>
       <securityConstraint httpMethods="GET POST" securedURLs="/mfp-oauth-java-tests/*"
          scope="Realm1 Realm2"/>
       <securityConstraint httpMethods="DELETE" securedURLs="/another-context-root/*"/>
   </OAuthTAI>
```

Set the security role

To protect the web resources used by your application, you must specify TAIUserRole as the Java Platform, Enterprise Edition (Java EE) security role. You can specify TAIUserRole as the Java EE security role in one of two ways: in the web.xml file or as an annotation.

To specify the TAIUserRole in the web.xml file, define TAIUserRole in the security-role element, and then use this role to secure the web resource in the security-constraint element. For example:

```
<security-constraint>
   <web-resource-collection>
      <web-resource-name>BaseServlet</web-resource-name>
      <url-pattern>/Test/devices</url-pattern>
   </web-resource-collection>
      <auth-constraint>
         <role-name>TAIUserRole</role-name>
      </auth-constraint>
</security-constraint>
<security-role id="SecurityRole_TAIUserRole" >
   <description>This is the role that MobileFirst OAuthTAI uses to protect the resource. It must be mapped to 'All Authenti
   <role-name>TAIUserRole</role-name>
</security-role>
```

To specify TAIUserRole with an annotation:

```
@WebServlet("/Test/devices")
@ServletSecurity(@HttpConstraint(rolesAllowed={"TAIUserRole"}))
public class BaseServlet extends HttpServlet {
    // servlet code ...
  }
```

Install and configure OAuthTAI in WebSphere Application Server or WebSphere Application Server Liberty

Installing OAuthTAI in WebSphere Application Server Liberty

1. Copy the file *product_install_dir*/WorklightServer/external-server-libraries/com.ibm.worklight.oauth.tai_1.0.0.jar to *usr*/extension/lib.

   where

product_install_dir
is the installation directory for MobileFirst Server.

usr is the user directory for theWebSphere Application Server Liberty profile (default name usr).

2. Copy the file OAuthTai-1.0.mf from the directory WorklightServer/external-server-libraries to the directory *usr*/extension/lib/features.
3. Edit the server.xml file to add the OAuthTAI -1.0 feature. For example:

```
<featureManager>
   <feature>appSecurity-2.0</feature>
   <feature>usr:OAuthTai-1.0</feature>
   <!--other necessary features-->
</featureManager>
```

4. Edit the server.xml file to add the OAuthTAI feature. Note that the security role TAIUserRole is mapped to a special subject named ALL_AUTHENTICATED_USERS. For example:

```
<usr_OAuthTAI id="myOAuthTAI">
   <securityConstraint httpMethods="GET POST" securedURLs="/worklight-oauth-java-tests/*"
       scope="Realm1 Realm2"/>
   <securityConstraint httpMethods="DELETE" securedURLs="/another-context-root/*"/>
</usr_OAuthTAI>

    <basicRegistry id="basic" realm="BasicRealm" />

    <application type="war" id="basicauth" name="basicauth" location="${server.config.dir}/apps/basicauth.war">
    <application-bnd>
    <security-role name="TAIUserRole">
     <special-subject type="ALL_AUTHENTICATED_USERS" />
     </security-role>
    </application-bnd>
   </application>
```

5. Add the URL to the authorization server into the environment variables of your deployment. One way of doing so is to add the entry into the server.env file, which is located in the same directory as the server.xml file. Add the following, where

```
 publicKeyServerUrl=URL_to_authorization_server.
```

URL_to_authorization_server is the URL to the MobileFirst authorization server.

Installing OAuthTAI in WebSphere Application Server

1. Copy *project_root_dir*/target/com.ibm.worklight.oauth.tai_1.0.0.jar to *product_install_dir*/lib/ext.
2. Open the WebSphere Application Server console.
3. Click **Security** > **Global Security** > **Authentication**, and select **Enable LTPA**.
4. Click **Security** > **Global Security** > **Application security** > **Enable application security.**
5. Configure the trust association.
   a. Click **Security** > **Global Security** > **Web and SIP security** > **Trust association**.
   b. Click **Enable trust association**.
   c. Under **Interceptors**, click **New**, and create a new interceptor with the following values:
      - **Interceptor class name**:com.worklight.oauth.tai.OAuthTAI
      - Custom properties: configFileLocation = *OAuthTAI_config_file_dir*/*filename*.xml

where

OAuthTAI_config_file_dir/filename.xml is the directory path and file name of the XML configuration file that you already created. See the configuration file example .

6. Create an environment entry that points to the authorization server.
   a. In the WebSphere Application Server console, click **Servers** > **Server Types** > **WebSphere application servers** > *your_server_name* > **Server Infrastructure** > **Java and Process Management** > **Process Definition** > **Additional Properties** > **Environment Entries**.
   b. Create the following environment variable:

```
publicKeyServerUrl=authorization_server_URL
```

where

authorization_server_URL
: is the URL to the MobileFirst authorization server.

7. Map the security role.
   a. In the WebSphere Application Server console, click **Applications** > **Application Types** > **WebSphere enterprise applications** > *your_application_name* > **Details Properties** > **Security role to user/group mapping**.
   b. Map the role that is allowed to access your application to **All Authenticated in Application's Realm**. For example: TAIUserRole as in sample above.
8. Restart WebSphere Application Server.

Add the tokens to the authorization header of your request

Add the following line to the authorization header of your request:

```
Authorization: Bearer  Access_token ID_token
```

where

Bearer (Mandatory) Is the required string for the token type, as defined in the OAuth 2.0 specification.

Access_token
(Mandatory) Encapsulates all of the security checks that the client has passed in the authorization phase.

ID_token
(Optional) Contains information about the user and device identity of the client.

When the application attempts to access resources protected by OAuth TAI, the following validation checks occur:

Validation item	Result if invalid
The authorization header exists.	A 401 error is returned, with the response header WWW-Authenticate: Bearer realm="imfAuthentication", scope="${required_scopes_list}"

Validation item	Result if invalid
The authorization header starts with the string `Bearer`.	A 400 error is returned, with the response header `WWW-Authenticate: Bearer realm="imfAuthentication", error="invalid_request", scope="${required_scopes_list}`
The signature of the access token and the ID token are valid and not expired, and theycan be decoded.	A 401 error is returned, with the response header `WWW-Authenticate: Bearer realm="imfAuthentication", error="invalid_request", scope="${required_scopes_list}"`
Validation of scope. This includes the following: • Verification of the scope in the access token. • Whether the mandatory scope is expired. • Whether the scope is expired. • Whether the access token contains all of the required realm names.	A 403 error is returned, with the response header `WWW-Authenticate: Bearer realm="imfAuthentication", error="insufficient_scope", error_description="${detail_information_of_error}.", scope="${required_scopes_list}"`

The security context object

After successful validation, a security context is available to your application. The security context is a JSON object that contains the following properties:

Subject: `securityContext.get("imf.sub");`
The subject is a string which uniquely identifies the application installation on a device and the authenticated user.

User Identity: (Optional)`securityContext.get("imf.user");`
The user identity is a JSON object which contains information about the authenticated user. For anonymous tokens, the user identity is null. The user identity contains the following properties:

- **"id"**: (Mandatory) The user identity of the authenticated user identity realm.
- **"authBy"**: (Mandatory):The name of the authenticated user identity realm.
- **"displayName"**: (Optional) The user's display name as set when authenticating the user identity realm. This property may be null.
- **"attributes"** : (Optional) Additional data that can be set when authenticating the user identity realm. This property may be null.

Application information:`securityContext.get("imf.application");`
Application information, including**"id"**, **"environment"**, and **"version"**. For example:

```
{"id":"com.MobileFirst.TestApp","environment":"iOSnative","version":"1.0"}
```

Device information:`securityContext.get("imf.device");`
Device information, including **"id"**, **"platform"**, **"model"** and **"osVersion"**. For example:

```
{"id":"E3D422A2-437C-438C-87FF-D6E7DF94725D","platform":"iOS","model":"iPhone Simulator","osVersion":"8.1"}
```

Example

```
  JSONObject securityContext = callerWLCredential.getSecurityContext();
  String subject = securityContext.get("imf.sub");
  JSONObject imfUser = securityContext.get("imf.user");
  JSONObject imfDevice = securityContext.get("imf.device");
  JSONObject imfApplication = securityContext.get("imf.application");
```

The WSCredential and WLCredential APIs provide credential functionality.

com.ibm.websphere.security.cred.WSCredential
: The WSCredential interface defines a credential that represents an authenticated principal to WebSphere Application Server or WebSphere Application Server Liberty. For example:

```
Subject callerSubject = WSSubject.getCallerSubject();
  WSCredential callerCredential = callerSubject.getPublicCredentials
     (WSCredential.class).iterator().next();
```

For more information about WSCredential, see the WSCredential documentation for your version of WebSphere Application Server or WebSphere Application Server Liberty. For example, see WSCredential for WebSphere Application Server Network Deployment 8.5.

com.worklight.oauth.tai.WLCredential
: The WLCredential interface provides APIs to get the MobileFirst specific principal details.

```
    WLCredential callerWLCredential = callerSubject.getPublicCredentials
          (WLCredential.class).iterator().next();
```

IBM MobileFirst Platform Foundation for iOS OAuthTAI in development mode

In development mode, you can copy the jar and the manifest file from the following locations, depending on your development environment:

MobileFirst Studio
: The root of every project contains a folder named `externalServerLibraries`. This folder contains both the TAI jar file (`com.ibm.worklight.oauth.tai_1.0.0.jar`) and the manifest file (`OAuthTai-1.0.mf`).

MobileFirst Platform Command Line Interface for iOS

The TAI jar file (`com.ibm.worklight.oauth.tai_1.0.0.jar`) and the manifest file (`OAuthTai-1.0.mf`) are located in the *`CLI_install_dir`*`/public` folder,

where

`CLI_install_dir`

is the path where MobileFirst Platform Command Line Interface for iOS is installed. Both files are also copied into the `/externalServerLibraries` folder in any new MobileFirst project created by the command-line interface.

Reporting analytics

The Token lib library can report analytic events to IBM MobileFirst Platform Operational Analytics. You must configure the Resource Server that contains the `Token lib` with your Analytics URL and credentials.

The MobileFirst Operational Analytics server is protected by basic authentication. When you installed this server, you configured the data entry point and basic authentication credentials. To configure the Resource Server, you must provide the Analytics credentials through JNDI properties, specifically the URL to the data entry point, the user name, and password. These properties are set with the following property names:

- `imf.analytics.url`
- `imf.analytics.username`
- `imf.analytics.password`

If your Resource Server is running on WebSphere Application Server Liberty, you must configure these properties by using JNDI by adding entries to your `server.xml` file. For example:

```
<jndiEntry jndiName="imf.analytics.username" value="admin"/>
```

If your Resource Server is running on WebSphere Application Server, you must configure these properties as Environment Entries. In the WebSphere Application Server Console, go to **Servers** > **Server Types** > **Websphere application servers** > **<*your server*>** > **Server Infrastructure** > **Java and Process Management** > **Process Definition** > **Additional Properties** > **Environment Entries** > **New**. This value is the location where you had to set `publicKeyServerUrl` when you set up OAuthTAI.

Protecting resources with the token validation endpoint:

The token validation endpoint on the IBM MobileFirst Platform Server validates tokens that are issued by the authorization server.

The token endpoint implements the OAuth 2.0 token introspection specification and validates access tokens and ID tokens. Using this endpoint, you can write a custom filter in any language to protect resources that are external to the MobileFirst Server. The filter delegates the token validation to the endpoint. An example of a custom filter in Node.js is shown at the end of this topic.

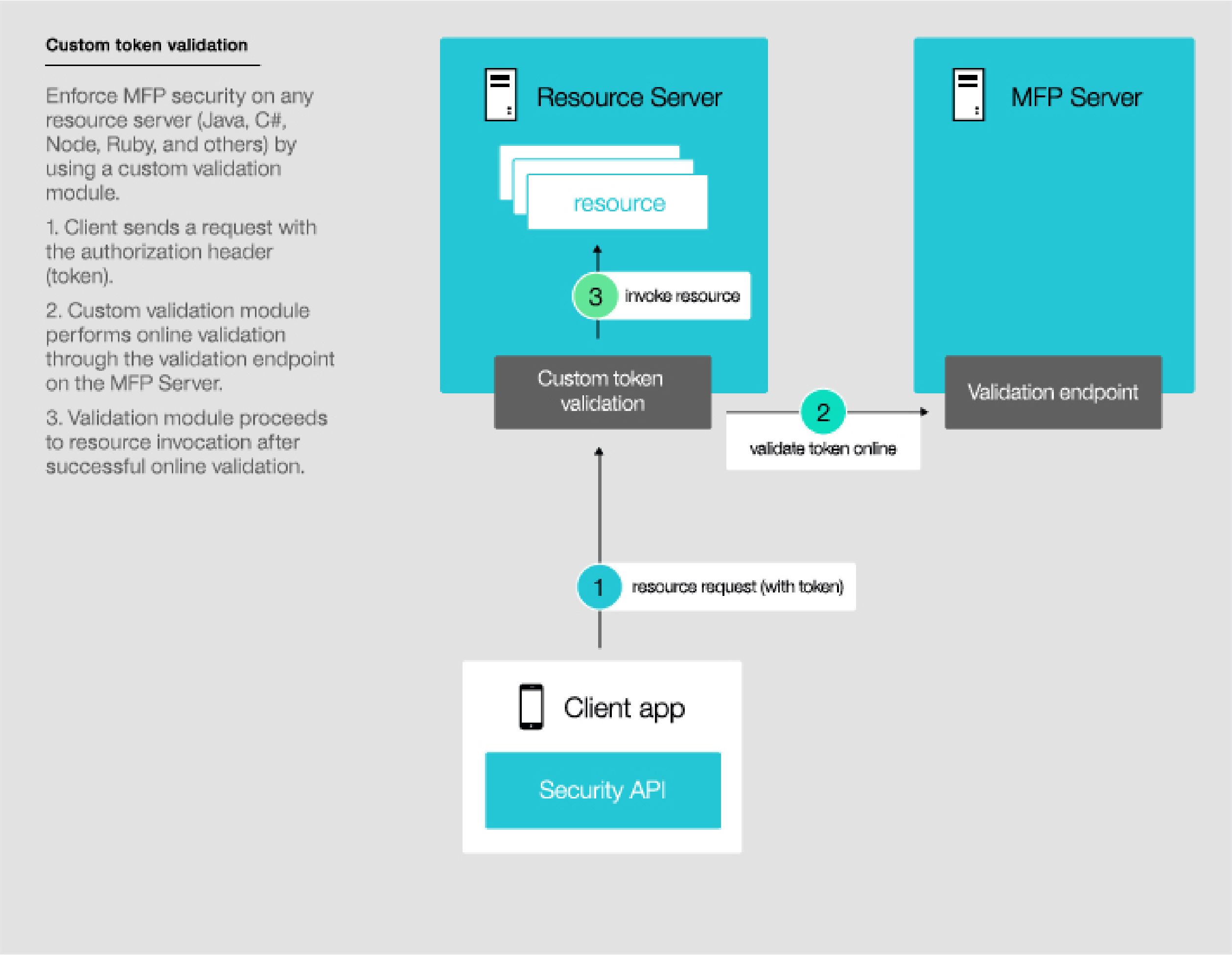

Custom token validation
Enforce MFP security on any resource server (Java, C#, Node, Ruby, and others) by using a custom validation module.
1. Client sends a request with the authorization header (token).
2. Custom validation module performs online validation through the validation endpoint on the MFP Server.
3. Validation module proceeds to resource invocation after successful online validation.
Resource Server
resource
3
invoke resource
Custom token validation
MFP Server
Validation endpoint
2
validate token online
1
resource request (with token)
Client app
Security API

Figure 8-35. Custom token validation

The endpoint has the same processing logic as the built-in Node.js filter and TAI filter.

Usage: The URL pattern for accessing the endpoint is as follows:

```
http(s)://<server_ip>:<server_port>/<project_name>/authorization/v1/token/validation
```

The endpoint responds to a POST request with the following parameters:

token Mandatory. The MobileFirst access token to be checked.

id_token
Optional. The MobileFirst ID token to be checked.

required_scope
Optional. If specified, the endpoint should ensure that the access token covers that scope.

Response: If the token validates successfully, the validation endpoint responds with a JSON object that is in `application/json` format and that has the following top-level members:

```
{
    "active": true,
    "exp" : timestamp,
    "scope" : "scope1 scope2 scope3",
    "user_id" : userid,
    "security_context" : securityContext,
}
```

active Indicates whether the token is valid or not.

exp Indicates the expiration time of the token.

scope Indicates the scopes of the token.

user_id
Indicates the unique identity of the token. If an ID token is provided, the value is a subfield of the ID token. If not, the value is the prn field in the access token.

security_context
Contains the decoded information from the access token and ID token. For more information, see The security context object.

In case of validation failure, the response conforms to the OAuth 2.0 specification, meaning that the status is 40* and WWW-Authenticate is added to the response header. The header looks like this:

```
WWW-Authenticate: Bearer realm="<realm name>", error="<error code>"[, error_description="<error description>"][, scope="<scope>"]
```

Example: The following example demonstrates a Node.js custom filter that uses the endpoint:

```
var express = require('express');
var http = require('http');
// For sending the data as x-www-form
var querystring = require('querystring');
var app = express();

// TokenValidationEndpoint custom filter
app.get('/resource/test0', function(req, res) {
    var authorizationHeader = req.headers["authorization"];
    var path = "/Top/authorization/v1/token/validation";
    var host = "9.148.225.200";
    var port = 10080;

    // Authorization header - Bearer  <accessToken> <idToken>
    var accessToken;
    var idToken;
    var tokenType;
    if (typeof authorizationHeader != 'undefined') {
        var tokenSplit = authorizationHeader.split(" ");
        tokenType = tokenSplit[0];
        accessToken = tokenSplit[1];
        idToken = tokenSplit[2];
    }
    var form = {
        token : accessToken,
        id_Token : idToken,
        token_type_hint : tokenType
    }

    var data = querystring.stringify(form);

    var options = {
        hostname : host,
        port : port,
        path : path,
        method : 'POST',
        headers : {
            'Content-Type' : 'application/x-www-form-urlencoded',
            'Content-Length' : Buffer.byteLength(data)
        }
    };

    var reqToWL = http.request(options, function(resFromWL) {
        resFromWL.on('data', function(chunk) {
            var parsedData = JSON.parse(chunk);
            if (!parsedData.active) {
                console.log("Token is invalid");
                var status = resFromWL.statusCode;
                var wwwAuthenticate = resFromWL.headers["www-authenticate"];
                // Set the header and send the response we got from the WL
                // server
                res.set("WWW-Authenticate", wwwAuthenticate);
                res.status(401).send();
            } else {
                console.log("Token is valid");
                // Send the response with the payload
                res.send("hello, user");
            }
        });
    });
    reqToWL.on('error', function(e) {
        console.log('problem with request to WL server: ' + e.message);
    });

    // write data to request body
    reqToWL.write(data);
    reqToWL.end();
});

app.listen(3000);
console.log("app is listening at " + 3000);
```

The test token endpoint

The test token endpoint enables you to get a valid token via a REST call, without having to set up a mobile client.

The OAuth model that is used in IBM MobileFirst Platform Foundation for iOS V7.0 requires a MobileFirst client API to be running in the mobile application that interacts with the MobileFirst Server to obtain an access token. Setting up the mobile app often means unnecessary overhead, particularly in back end testing situations. Use the test token endpoint to get a valid token through a REST call (by using tools such as Postman or cURL), without a mobile client.

Note: The test token endpoint is available only in the development version of IBM MobileFirst Platform Foundation for iOS.

Usage

The endpoint provides a valid token with a default expiration of two hours and a scope that includes all the realms that are defined in your `authenticationConfig.xml` file.

The URL pattern for accessing the endpoint is as follows:

`http(s)://<server_ip>:<server_port>/<project_name>/authorization/v1/testtoken`

The endpoint responds to a POST request with the following parameters:

accessTokenExpiration
Number of seconds for the token expiration.

tokenFormat: Header/Token

Response format:

1. Header (default): returns a JSON object that can be copied directly to REST apps (such as Postman) as a header, for example,

   `{Authorization: Bearer eyJhbG...}`
2. Token: returns the OAuth token as defined by the spec:

```
{
    "scope": "SubscribeServlet wl_directUpdateRealm wl_authenticityRealm SampleAppRealm wl_remoteDisableRealm wl_antiXSR
    "token_type": "bearer",
    "expires_in": 1421262002284,
    "id_token": "eyJhbG...",
    "access_token": "eyJhb.."
}
```

Exposing mobile services to non-mobile (confidential) clients

Using IBM MobileFirst Platform Foundation for iOS you can let a confidential (or non-mobile) client connect to mobile services in a secure way. For example, you can grant a back-end service access to the Push service.

Overview

The confidential client flow is based on the client credentials flow, as defined in the OAuth specification. The client authenticates with the authorization server using a certificate. The authentication server validates the certificate, and if it is valid, issues an access token to the client. The certificate has to be signed by a trusted signer certificate that is configured at the authorization server.

Structure of the client certificate

The confidential client certificate contains three attributes that are required for the token generation:

- **Client ID**: a string that identifies the confidential client, that is, the certificate holder. The client ID is used to identify the client, for example for collecting analytics.
- **Scope**: a list of scope tokens (realm names) that can be granted to the certificate holder. Unlike realms that are used by mobile clients, the realms for certificate holders do not have to be defined in the `authenticationConfig.xml` security configuration file of the application. Built-in MobileFirst realms and realms that are defined in the security configuration file may also be included.
- **Expiration**: the expiration time (in seconds) of tokens that are generated for the certificate holder.

The Client ID attribute is stored as the common name attribute of the certificate. The Scope and Expiration attributes are stored as X.509 certificate extensions. A sample script has been provided that generates a certificate with these three attributes, and signs it with the trusted signer certificate. For more information, see Generating the trusted signer certificate.

Acquiring tokens

To acquire a token, the client authenticates itself with the authorization server by providing a JSON Web Signature (JWS). The payload of the JWS contains a timestamp and is signed by the certificate of the confidential client. The signature is valid for one minute only, meaning that clients must generate a new signature with every token request.

The client posts a token request to the authorization server's token endpoint. The request includes the signature that allows the server to authenticate the client. The request might also include a Scope parameter that specifies the scope to be granted in the token. The Scope parameter has to be a subset of the scope that is defined in the certificate of the client. If no scope is included in the request, the result token is an access token that includes no realms (but this is still a valid token).

If the signature is verified correctly, the server responds with an access token. The access token is generated using the attributes that are specified in the certificate - the client ID, the token expiration, and the scope (or a subset of the scope if specified in the scope parameter of the token request). The expiration time of the individual realms is the same as the token expiration time.

The sample application, Confidential_Client_Sample can be used for constructing a signature, sending a request to the token endpoint, and extracting the access token from the response. Download Confidential_Client_Sample.

Generating a client certificate

A sample script has been provided for generating a certificate for a confidential client. The script puts the custom attributes (Client ID, Scope, and Expiration) in the certificate and signs the certificate with the trusted signer certificate. The script uses OpenSSL, so ensure that you have OpenSSL installed. The script is called `gen_cert.sh`.

The `gen_cert.sh` script requires three parameters:

- **Workdir**: the path of the working directory. The script uses two files that must be present in the working directory: rootCA.crt, the trusted signer certificate, and rootCA.key, the private key of the trusted signer certificate.
- **clientId**: the client ID attribute to be stored in the certificate.
- **password**: the password for protecting the private key of the client certificate and the resulting KeyStore.

The scope and expiration attributes to be placed in the certificate are taken from the configuration file `extensions.cnf` that is located in the directory of the script. Before running the script, edit this file and set the values for scope and expiration.

The script generates the following files:

- **`certificate.crt`**: the client certificate.
- **`private_key.key`**: the private key of the client certificate.
- **`keystore.p12`**: a PKCS #12 KeyStore that stores the client certificate and the private key.

Download the Certificate_Scripts sample script.

Generating the trusted signer certificate

You can generate the trusted signer certificate by creating a certificate-signing request and submitting it to a chosen certificate authority. The trusted signer certificate may also be self-signed. A sample script, `gen_root_cert.sh` generates a self-signed signer certificate. It requires a single parameter - the path of the output directory. The script generates two files: `rootCA.crt`, the signer certificate, and `rootCA.key`, the private key of the certificate. The default password is `abcdef`, but it can be changed by editing the script. The script is included in Certificate_Scripts.zip.

Configuring the trusted signers

The list of trusted signer certificates is configured in the property trusted.signer.certificate.paths. The list is space-separated. The paths may be relative to the server folder in the MobileFirst Project (for example: conf/rootCA.crt), or absolute paths.

Sample client project

You can download Confidential_Client_Sample, as a zip file. The sample project contains code for obtaining an access token by generating a JSON web signature, and submitting a request to the token endpoint. The sample project contains a KeyStore file, keystore.p12 that was generated with the sample script gen_cert.sh. The KeyStore contains a client certificate that is signed with the sample signer certificate (rootCA.crt, located in the zip file). The scope attribute of the client certificate has the value scope1 scope2 scope3, and the sample code shows a request for a token with scope scope1, which is a subset of the scope that is defined in the certificate. To run the sample, perform the following steps:

1. Configure the authorization server to use the root certificate rootCA.crt as a trusted signer. Put the certificate file rootCA.crt inside your MobileFirst Platform project and set the value of the property trusted.signer.certificate.paths accordingly. For example, if you put the root certificate in the conf directory of your MobileFirst Platform project, the property value is trusted.signer.certificate.paths=conf/rootCA.crt.
2. Edit the constant MFP_PROJECT_URL in the source file ConfidentialClientSample.java, and replace the host, the protocol, and the MobileFirst Platform project name with the appropriate values. Run the ConfidentialClientSample Java application. The application requests an access token from the authorization server and prints it to the console.

Custom requests to resources using Objective-C

This sample illustrates how to get data from a protected resource by using a custom NSURLRequest and the MobileFirst AuthorizationManager API.

Inside the didReceiveResponse delegate, the authorizationManager determines whether this response is a MobileFirst protocol response. If so, the user gets the scope, obtains the authorization header for this scope, and requests the protected resource one more time.

```
-(IBAction)sendCustomRequestToResource:(id)sender {
  // The URL of the resource with filter
  NSString *nodeResourceString = @"http://localhost:3000/v1/apps/1234/test";

  // Create the first request to the resource - will fail due to authentication.
  NSURLRequest *request = [NSURLRequest requestWithURL:[NSURL URLWithString:nodeResourceString]];

  // Create url connection and fire request with a delegate
  NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:request delegate:self];

}

-(void) connection:(NSURLConnection *)connection didReceiveResponse:(NSURLResponse *)response{
  WLAuthorizationManager *authorizationManager = [WLAuthorizationManager sharedInstance];

  // Check that the response conforms to the MFP protocol
  if([authorizationManager isAuthorizationRequiredForResponse:response]){
    // Get required scope from response
    NSString *scope = [authorizationManager authorizationScopeForResponse:response];

    //Obtain authorization header for the scope
    [[WLAuthorizationManager sharedInstance] obtainAuthorizationHeaderForScope:scope
completionHandler:^(WLResponse *wlResponse, NSError *wlError) {
      if (!wlError) {
        // Create the request.
        NSMutableURLRequest *mutRequest = [NSMutableURLRequest
requestWithURL:[NSURL URLWithString:@"http://localhost:3000/v1/apps/1234/test"]];

        // Add the Authorization header that was obtained
        [mutRequest addValue:authorizationManager.cachedAuthorizationHeader forHTTPHeaderField:@"authorization"];
        NSError *requestError; // Error from resource (if any) will be here

        // Create URL connection and fire request with a delegate
        NSURLConnection *conn = [[NSURLConnection alloc] initWithRequest:mutRequest delegate:self];

      } else {
        // Error obtaining Authorization header from MFP Server
        NSLog(@"%@",[wlError localizedDescription]);
      }
    }];
  }
  else{
    //This is the applicative response from the resource
    NSLog(@"Meta-data response from resource: %@",[response description]);
  }
}
```

HTTP-based custom authentication

Custom authentication is a form of API authentication and is based on an HTTP protocol.

Overview

The custom identity provider authenticates a user by sending challenges to the client. However, custom identity providers do not communicate directly with clients. They send challenges and receive responses to the challenges by means of the IBM MobileFirst Platform Foundation for iOS authorization server. The process of authenticating a user typically involves a dialog with the client and consists of several requests and responses.

The HTTP protocol that is used in custom authentication is between IBM MobileFirst Platform Foundation for iOS authorization server and custom code. The custom code can be written in any language and can be protected or unprotected. If you choose to protect it, implement protection by using the default scope. For example, the Node.js sample on this page uses:

```
passport.authenticate('mobilefirst-strategy', {
    session : false
}
```

to protect the startAuthorization resource.

To implement custom identity provider, you must expose the following two HTTP endpoints:

- /startAuthorization endpoint. The headers that it receives are those which the client sends to the IBM MobileFirst Platform Foundation for iOS and which the server then adds to the request body. For example, the Node.js sample on this page gets the headers by using the following code:

  ```
  function(req, res) {
      var returnedJSON = startAuthorization(req.body.headers);
      res.json(returnedJSON);
  }
  ```

- /handleChallengeAnswer endpoint. The headers that it receives are those which are received at the /startAuthorization endpoint, with the addition of an answer to a challenge and a state ID. This HTTP endpoint checks the challenge and returns `success`, `challenge`, or `failure`.

The custom identity provider may store some state related to the authentication dialog. An example use case is multi-step authentication, where the custom identity provider needs to store the result of the first authentication step when proceeding to the next step. In this case, the dialog between IBM MobileFirst Platform Foundation for iOS authorization server and the custom identity provider becomes stateful. The custom identity provider assigns a state ID, which is sent in the response. The IBM MobileFirst Platform Foundation for iOS authorization server passes the state ID in subsequent requests that belong to the client authentication process.

Headers are sent in the body because it is the header from the client request. Sending headers from the client in the body from authorization server to the custom identity provider will prevent conflicts in header names. In case you need to save state between calls to startAuthorization and handleChallengeAnswer, use the optional `stateId` property in the challenge answer. The custom identity provider and the protected resource may be implemented on the same server or on different servers.

The realm is always in the URL path, just before the endpoint API, as shown in the following examples:

- **`http://myhost:myport/custom_realm/startAuthorization`**
- **`http://myhost:myport/custom_realm/handleChallengeAnswer`**

Both HTTP endpoints can return three types of JSON response:

- **challenge**

```
{
status: "challenge", challenge:
    { your custom JSON challenge },
    stateId : custom state id //state id is optional
}
```

- **success**

```
{
 status: "success", userIdentity:
    {userName: userName, displayName: displayName, attributes: { your optional custom JSON attr
}
```

- **failure**

```
{ status: "failure" }
```

Samples

Let's say you want to protect an HTTP resource that is defined by the URL /protected_resource, with the scope "customAuthRealm_1 customAuthRealm_2". Perform the following steps:

1. Create an IBM MobileFirst Platform Foundation for iOS project.
2. In the project, define two realms and their login modules in the `authenticationConfig.xml` file.

   Note: The `className` property that is shown both in the realm and login module for this example must be used **as is, in all cases**.

```
<realms>
    <realm name="customAuthRealm_1" loginModule="customAuthLoginModule_1">
        <className>com.worklight.core.auth.ext.CustomIdentityAuthenticator</className>
        <parameter name="providerUrl" value="http://localhost:3000"/>
    </realm>

    <realm name="customAuthRealm_2" loginModule="customAuthLoginModule_2">
        <className>com.worklight.core.auth.ext.CustomIdentityAuthenticator</className>
        <parameter name="providerUrl" value="http://localhost:3000"/>
    </realm>
</realms>
<loginModules>
    <loginModule name="customAuthLoginModule_1">
        <className>com.worklight.core.auth.ext.CustomIdentityLoginModule</className>
    </loginModule>

    <loginModule name="customAuthLoginModule_2">
        <className>com.worklight.core.auth.ext.CustomIdentityLoginModule</className>
    </loginModule>
</loginModules>
```

3. Configure the realm to be a user identity realm: To get the user identity in Java adapter, Node.js, or TAI resource, you must specify them in the `application-descriptor.xml` file of the application, in the `user identity realms` field.

   Note: User identity realms are comma separated. The order of the realms dictates the selected user identity. If the list is empty, the ID token contains no identity information.
4. Now proceed to the Node.js or Java JAX-RS sample code in the next sections to see a full implementation of the solution.

Node.JS sample

The following sample code demonstrates how to implement the startAuthorization and handleChallengeAnswer endpoints in Node.js.

```
// Call the protected_nodejs_resource
    app.get('/protected_nodejs_resource',
      passport.authenticate('mobilefirst-strategy', {session : false, scope: "customAuthRealm_1 custo
       res.status(200).send("Hello capella from node.js"));
    });

//Implement custom authentication for customAuthRealm_1
    app.post('/customAuthRealm_1/startAuthorization',
        passport.authenticate('mobilefirst-strategy', {
            session : false
        }),
        function(req, res) {
            var returnedJSON = startAuthorization(req.body.headers);
            res.json(returnedJSON);
    });

    app.post('/customAuthRealm_1/handleChallengeAnswer',
            passport.authenticate('mobilefirst-strategy', {
                session : false
            }),     function(req, res) {
            var returnedJSON = handleChallengeAnswer(req.body.headers, req.body.stateId, req.body.cha
            res.json(returnedJSON);
    });


    var startAuthorization = function(headers) {
        return {
            status: "challenge",
                challenge: {
                    message: "missing_credentials"
                },
            stateId : "my_custom_state_id"
        };
    };


    var handleChallengeAnswer = function(headers, stateId, challengeAnswer) {
        if (challengeAnswer && users[challengeAnswer.userName] && challengeAnswer.password === use
            return {
                status: "success",
                userIdentity: {
                    userName: challengeAnswer.userName,
                    displayName: users[challengeAnswer.userName].displayName,
                    attributes : {"customAttr1":"customValue1", "customAttr2":"customValue2"}

                }
            };
        } else {
            return {
                status: "failure"
            }
        }
    };
```

Java JAX-RS sample

The following JAX-RS sample is implemented as a Java REST adapter. The startAuthorization and handleChallengeAnswer endpoints have been implemented in Java.

```
// Call the protected_java_resource
@GET
@Path("/protected_java_resource")
@Produces("application/json")
@OAuthSecurity (scope="customAuthRealm_1 customAuthRealm_2")
public JSONObject helloCapella(){
    JSONObject hello = new JSONObject();
    hello.put("hello", "Hello Capella");
    return hello;
}



@POST
    @Consumes ("application/json")
    @Path("/customAuthRealm_1/startAuthorization")
    @Produces(MediaType.APPLICATION_JSON)
    public JSONObject startAuthorization(String payload) throws Exception{
        logger.info(payload);
        JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
        return returnJson;
    }


   @POST
   @Consumes ("application/json")
   @Path("/customAuthRealm_1/handleChallengeAnswer")
   @Produces(MediaType.APPLICATION_JSON)
    public JSONObject handleChallengeAnswer(String payload) throws Exception{
        JSONObject userStoreJson = (JSONObject) JSON.parse(USER_STORE_JSON);
        JSONObject failedResponseJson = (JSONObject) JSON.parse(FAILURE_JSON);

        if(payload == null || payload.isEmpty()) {
            return failedResponseJson;
        }
        JSONObject payloadJson = (JSONObject) JSON.parse(payload);
        JSONObject challengeAnswer = (JSONObject) payloadJson.get("challengeAnswer");

        if (challengeAnswer == null ) {
            return failedResponseJson;
        }

        String userName = (String) challengeAnswer.get("userName");
        String password = (String) challengeAnswer.get("password");

        if (userName == null || userName.isEmpty() || password == null || password.isEmpty()) {
            return failedResponseJson;
        }

        if (userStoreJson.containsKey(userName)) {
            JSONObject userInfoJson = (JSONObject) userStoreJson.get(userName);
            String userPassword = (String) userInfoJson.get("password");
            String userDisplayName = (String) userInfoJson.get("displayName");

            if (password.equals(userPassword)) {
                JSONObject returnJson = new JSONObject();
                JSONObject userIdentityJson = new JSONObject();
                userIdentityJson.put("userName", userName);
                userIdentityJson.put("displayName", userDisplayName);
                JSONObject customAttributes = new JSONObject();
                customAttributes.put("custom1Key", "custom1Value");
                customAttributes.put("custom2Key", "custom2Value");
                userIdentityJson.put("attributes", customAttributes);
                returnJson.put("status", "success");
                returnJson.put("userIdentity", userIdentityJson);
                return returnJson;
            }
        }

        return failedResponseJson;
    }
```

For more information

More techniques for implementing custom authentication

IBM MobileFirst Platform Foundation for iOS provides other custom authentication mechanisms:

- Adapter-based custom authentication. For more information, see Implementing adapter-based authenticators.
- Java-based custom authentication. For more information, see see Implementing Java-based custom authenticators.

Classic security model

Topics that describe classic (pre-V7.0) security features in IBM MobileFirst Platform Foundation for iOS.

The following sections provide high-level information about the MobileFirst security model.

Goals and structure of MobileFirst security framework

The MobileFirst security framework serves two main goals. It controls access to the protected resources, and it propagates the user (or server) identity to the back-end systems through the adapter framework.

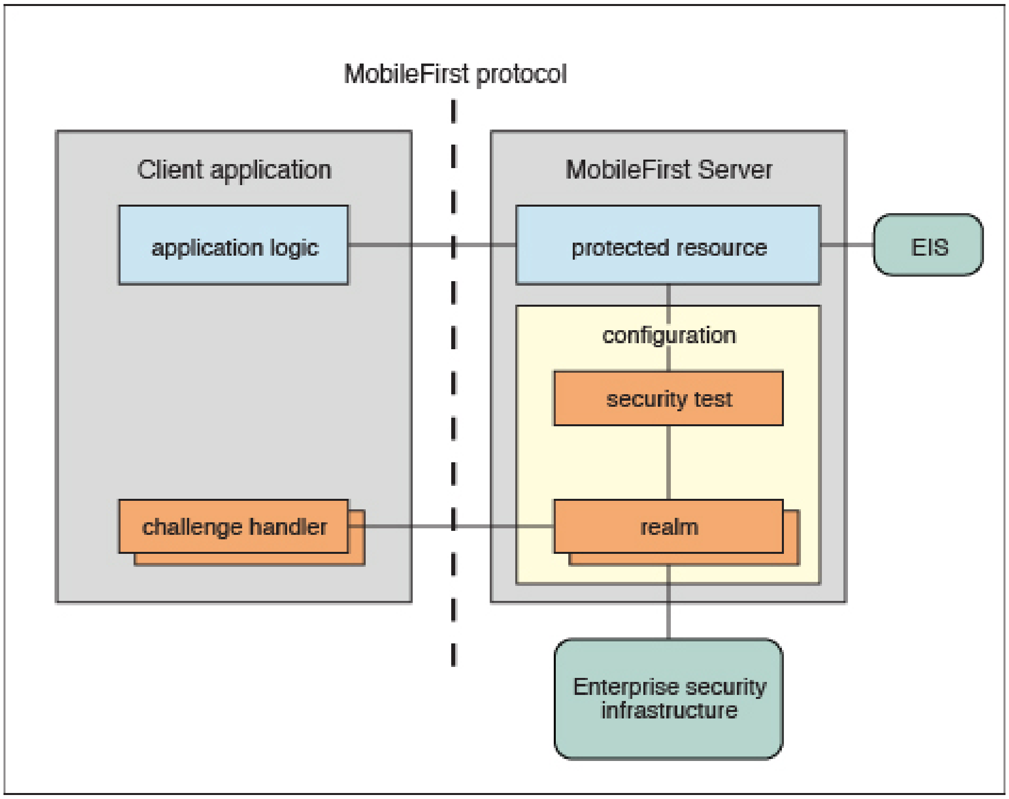
It is key to the success of the application that the MobileFirst security framework does not include its own user registry, credentials storage, or access control management. Instead, it delegates all those functions to the existing enterprise security infrastructure. This delegation allows MobileFirst 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 MobileFirst security framework, and supports custom extensions that allow integration with virtually any security mechanism.

Another feature of the IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS security framework provides a *wire protocol*. This protocol allows the combination of challenges and responses of multiple security checks during a single request-and-response round trip. The protocol serves two important purposes: it allows the number of extra round trips between the client and server to be minimized, and it separates the application logic and the security checks implementation.



Protected resources and authentication context

A protected resource can be any of the following items:

- **Application**

  Any request to the application requires successful authentication in all realms of the security test that is defined in the application descriptor.

- **Adapter procedure**

  Procedure invocation requires successful authentication in all realms of the security test that is defined in the adapter descriptor. The user identity and credentials that are obtained during such authentication can be propagated to the enterprise information system represented by this adapter.

- **Event source**

  Subscription to push notifications requires successful authentication in all realms of the security test that are defined in the event source definition (in adapter JavaScript).

- **Static resource**

  Static resources are defined as URL patterns in the authentication configuration file. They allow protection of "static" web applications such as the MobileFirst Operations 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 MobileFirst Server
- Direct update
- Anti-XSRF check (cross-site request forgery)

The realms are defined in the authentication configuration file on the MobileFirst 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 MobileFirst Platform Foundation for iOS provides simplified security test configurations for mobile and web environments. It is also possible to create a custom security test from scratch.

MobileFirst 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, MobileFirst 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.

MobileFirst 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 MobileFirst 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 MobileFirst client infrastructure resends the original request with all the challenge responses.

MobileFirst Server extracts those responses from the request and passes them to the appropriate authenticators. If an authenticator is satisfied, it reports a success, and MobileFirst 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, MobileFirst 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 MobileFirst 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

MobileFirst 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 MobileFirst Server, and vice versa. Since this task can be non-trivial, it is important to understand the differences between IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Server and the Java Platform, Enterprise Edition container is implemented as a custom IBM MobileFirst Platform Foundation for iOS realm. This realm can interact with the container and obtain and set its authenticated principal.

MobileFirst 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 MobileFirst user identity. This case assumes that the MobileFirst WAR file has appropriate login configuration and resource collection definitions. Including this information can be tricky because the `web.xml` file for MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst Server. This sequence implies the following requirements on the MobileFirst 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 MobileFirst Platform Foundation for iOS, 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 MobileFirst Platform Foundation for iOS to provide the following protections by using risk-based access decisions to protect MobileFirst 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 MobileFirst Platform Foundation for iOS and application authentication.

For more information about IBM Security Access Manager, see IBM Security Access Manager for IBM MobileFirst Platform Foundation for iOS.

MobileFirst application authenticity overview

An overview of application authenticity features and procedures within IBM MobileFirst Platform Foundation for iOS

IBM MobileFirst Platform Foundation for iOS framework provides a number of security mechanisms. One of them is a security test for application authenticity. Most IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Server is authentic and was not tampered with or modified by some attacker.

Starting with IBM MobileFirst Platform Foundation for iOS V7.0, you can enable one of three levels of authentication for your app: none, basic, and extended. For more information about enabling extended app authentication, see “Configuring extended app authenticity checking” on page 11-56. Extended application-authenticity validation is not supported for iOS apps that are installed from the Apple app store. See the Extended application authenticity for iOS known limitation.

Authenticity process

Application authenticity checks use the same transport protocol as other authentication framework realms:

1. The application makes an initial request to MobileFirst Server.
2. MobileFirst Server goes through the authentication configuration and finds that this application must be protected by an application authenticity realm.
3. MobileFirst 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 MobileFirst Server.
7. If the challenge response is valid, MobileFirst Server serves the application with the required data.
8. If the challenge response is invalid, MobileFirst Server refuses to serve the application.

Authenticity considerations during migration to MobileFirst Server V7.0

In versions earlier than MobileFirst Platform Foundation V7.0, there was an option to control application authenticity from the MobileFirst Operations Console. This option is no longer available starting from V7.0. Existing applications configured with an authenticity realm and a security test in `authenticationConfig.xml` and a security test and security attributes in the `application-descriptor.xml` file are checked for authenticity at runtime when migrated to MobileFirst Server V7.0, ignoring any previous authenticity control mode that was specified in an older version of MobileFirst Operations Console.

Enabling an application authenticity check (example)

The following example shows the steps for enabling application authenticity on iOS:

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."
3. After you have updated the required elements, rebuild and redeploy your application to MobileFirst Server.

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 MobileFirst Platform Foundation for iOS checks whether the client is already authenticated according to all realms of the security test. If the client is not yet authenticated, IBM MobileFirst Platform Foundation for iOS 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.

Note: To replace an existing security test, you should add the new security test and deploy it, after this new security test is successfully deployed you can delete the old security test that you intended to replace.

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 MobileFirst Operations Console, the ability for the app to connect to MobileFirst 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

This section describes what a webSecurityTest and a mobileSecurityTest contain.

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_directUpdateRealm, 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>
```

Security configurations of protected resources:

There is a default security configuration for every type of resource that can be protected by the MobileFirst authentication framework.

Protected resource types

The following types of resource can be protected by using the MobileFirst authentication framework:

Application environment

With IBM MobileFirst Platform Foundation for iOS, you can define a separate security test for each application environment. This capability is very useful, as different environments support different sets of security features. For example, BlackBerry and WindowsPhone environments do not support application authenticity test. In case no security test is explicitly defined, each application environment is protected by a relevant default type of security test. Mobile hybrid and native environments are protected by a default mobileSecurityTest and web environments are protected by a default webSecurityTest. If you want to protect application environments with a security test other than the default one, you must explicitly specify it in the `application-descriptor.xml` file. For example:

```
<android version="1.0" securityTest="mobileTests">
    <worklightSettings include="false"/>
    <security>
        <encryptWebResources enabled="false"/>
        <testWebResourcesChecksum enabled="false" ignoreFileExtensions="png, jpg, jpeg, gif, mp4, mp3"/>
        <publicSigningKey/>
        <packageName/>
    </security>
</android>
<mobileWebApp cacheManifest="no-use" securityTest="webTests"/>
```

Adapter procedure

With IBM MobileFirst Platform Foundation for iOS, you can protect any adapter procedure by using a security test. Doing so will ensure that the information that is provided by adapters is exposed to the client side only after successful completion of all the relevant authentication steps. Because an adapter procedure is a service that is provided by MobileFirst Server, it is protected with a default `webSecurityTest` security test. To protect adapter procedures with a security test that is different from the default one, you need to set the **`securityTest`** attribute of the relevant <procedure> element in the adapter-descriptor XML file. See "Structure of the adapter XML file" on page 8-38. The following example protects the `myAccounts` procedure with an `adapterProcedureSecurityTest` security test:

```
<procedure name="myAccounts" securityTest="adapterProcedureSecurityTest"/>
```

Note: This protection method of the classic security model is applicable to JavaScript adapter procedures. Java JAX-RS adapter procedures are protected by using the @OAuthSecurity annotation, which is part of the

Java server-side com.worklight.core.auth package. See the annotation's API reference, as well as “Security configuration of a JAX-RS resource” on page 8-30.

EventSource

Using an event source, you can subscribe to server-generated events, for example, push notifications or SMS notifications. Subscription is based on user identity and device identity, because notifications are usually sent to either a specific device or to a specific user. So, it is important to specify security test in your event source declaration. The user and device identities that are retrieved as specified in a security test are used to subscribe to events. To define a security test for event source, add a securityTest property to event source declaration code.

```
WL.Server.createEventSource({
    name: 'PushEventSource',
    onDeviceSubscribe: 'deviceSubscribeFunc',
    onDeviceUnsubscribe: 'deviceUnsubscribeFunc',
    securityTest: 'eventSourceSecurityTest'
});
```

Static resource

A static resource is a URL that can be accessed on MobileFirst Server. In most cases, the URL is of the MobileFirst Console. Static resources protection can be defined in the <staticResources> section of the `authenticationConfig.xml` file. There is no default security test for static resources, so MobileFirst Console is not protected, by default. The developer must explicitly specify the static resource to protect in the `authenticationConfig.xml` file and set the desired securityTest.

```
<staticResources>
    <resource id="worklightConsole" securityTest="worklightConsole">
        <urlPatterns>/console*</urlPatterns>
    </resource
</staticResources>
```

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 MobileFirst Server together with your public key infrastructure (PKI).

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 MobileFirst Platform Foundation for iOS 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 MobileFirst Server receives an HTTP 401 ("Unauthorized") response that contains the `WL-Instance_Id` token. The MobileFirst 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.

Note: If code that uses the anti-XSRF realm attempts to access a resource that is protected by OAuth authentication, and the client has a valid token, the MobileFirst Server is not called. As a result, the server does not check whether the request contains the header. The MobileFirst Server is called when the token expires or when the anti-XSRF realm inside the token expires. When the realm expires, the anti-XSRF authenticator is invoked and the server checks whether the request contains the header. For more information about OAuth authentication, see MobileFirst OAuth-based security model.

Certificate pinning

Use certificate pinning to help prevent man-in-the-middle attacks.

When communicating over public networks it is essential to send and receive information securely. The protocol widely used to secure these communications is SSL/TLS. (SSL/TLS refers to Secure Sockets Layer or to its successor, TLS, or Transport Layer Security.) SSL/TLS uses digital certificates to provide authentication and encryption. To trust that a certificate is genuine and valid, it is digitally signed by a root certificate belonging to a trusted certificate authority (CA). Operating systems and browsers maintain lists of trusted CA root certificates so that they can easily verify certificates that the CAs have issued and signed.

Protocols that rely on certificate chain verification, such as SSL/TLS, are vulnerable to a number of dangerous attacks, including man-in-the-middle attacks, which occur when an unauthorized party is able to view and modify all traffic passing between the mobile device and the backend systems.

IBM MobileFirst Platform Foundation for iOS provides an API to enable certificate pinning. It is supported in native iOS, native Android, and hybrid iOS or hybrid Android MobileFirst applications.

Certificate pinning process

Certificate pinning is the process of associating a host with its expected public key. Because you own both the server-side code and the client-side code, you can configure your client code to accept only a specific certificate for your domain name, instead of any certificate that corresponds to a trusted CA root certificate recognized by the operating system or browser.

A copy of the certificate is placed in your client application. During the SSL handshake (first request to the server), the IBM MobileFirst Platform Foundation for iOS client SDK verifies that the public key of the server certificate matches the public key of the certificate that is stored in the app.

Important:

- Some mobile operating systems might cache the certificate validation check result. Therefore, your code should call the certificate pinning API before making a secured request. Otherwise, any subsequent request might skip the certificate validation and pinning check.
- Make sure to use only MobileFirst APIs for all communications with the related host, even after the certificate pinning. Using third-party APIs to interact with the same host might lead to unexpected behavior, such as caching of a non-verified certificate by the mobile operating system.
- Calling this method a second time overrides the previous pinning operation.

If pinning is successful, the public key inside the provided certificate is used to verify the integrity of the MobileFirst Server certificate during the secured request SSL/TLS handshake. If pinning fails, all SSL/TLS requests to the server are rejected by the client application.

Certificate setup

You must use a certificate purchased from a certificate authority. Self-signed certificates are not supported. For compatibility with the supported environments, make sure to use a certificate that is encoded in DER (Distinguished Encoding Rules, as defined in the International Telecommunications Union X.690 standard) format.

You must place the certificate in both the MobileFirst Server and in your application. Place the certificate as follows:

1. In the MobileFirst Server: (WebSphere Application Server, WebSphere Application Server Liberty, or Apache Tomcat). Consult the documentation for your specific application server for information about how to configure SSL/TLS and certificates.
2. In your application, add the certificate to the application bundle.

Certificate pinning API

Certificate pinning consists of a single API method, that has a parameter *certificateFilename*, where *certificateFilename* is the name of the certificate file.

Native iOS

Syntax:

```
pinTrustedCertificatePublicKeyFromFile:(NSString*) certificateFilename;
```

Example:

```
[[WLClient sharedInstance]pinTrustedCertificatePublicKeyFromFile:@"myCertificate.cer"];
```

The certificate pinning method will raise an exception in two cases:

- The file does not exist
- The file is in the wrong format

Note: Certificate pinning is not supported for Cordova applications.

For more details on the certificate pinning API, see the following reference sections:

- Native iOS: Objective-C client-side API WLClient class.

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 MobileFirst application on the device and the MobileFirst 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 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, IBM MobileFirst Platform Foundation for iOS 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, IBM MobileFirst Platform Foundation for iOS internally signs certificates and manages the validation and enrollment process.

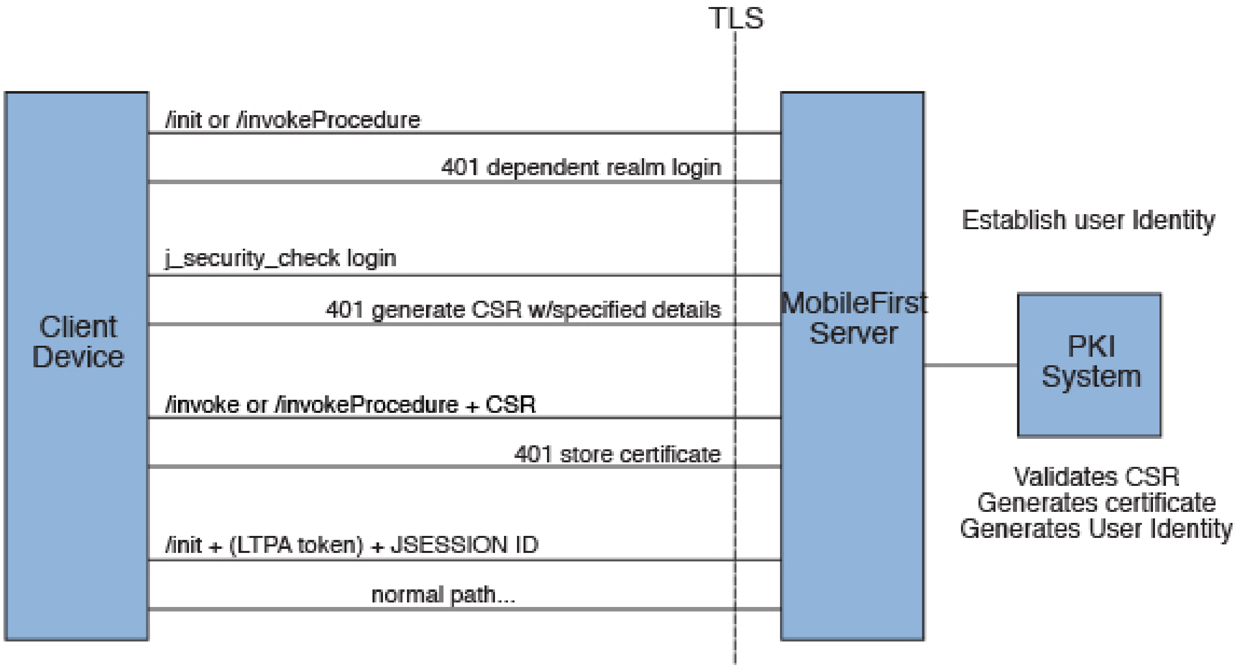
More specifically, mobile clients are now able to present an X.509 client certificate to establish a secure connection over the transport layer security (TLS) protocol. Users are enrolled to the enterprise certificate authority (CA) directly from their device. The client certificate is then used to authenticate and establish a user identity on subsequent requests.

How it works

The MobileFirst 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 IBM MobileFirst Platform Foundation for iOS 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, IBM MobileFirst Platform Foundation for iOS, 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:

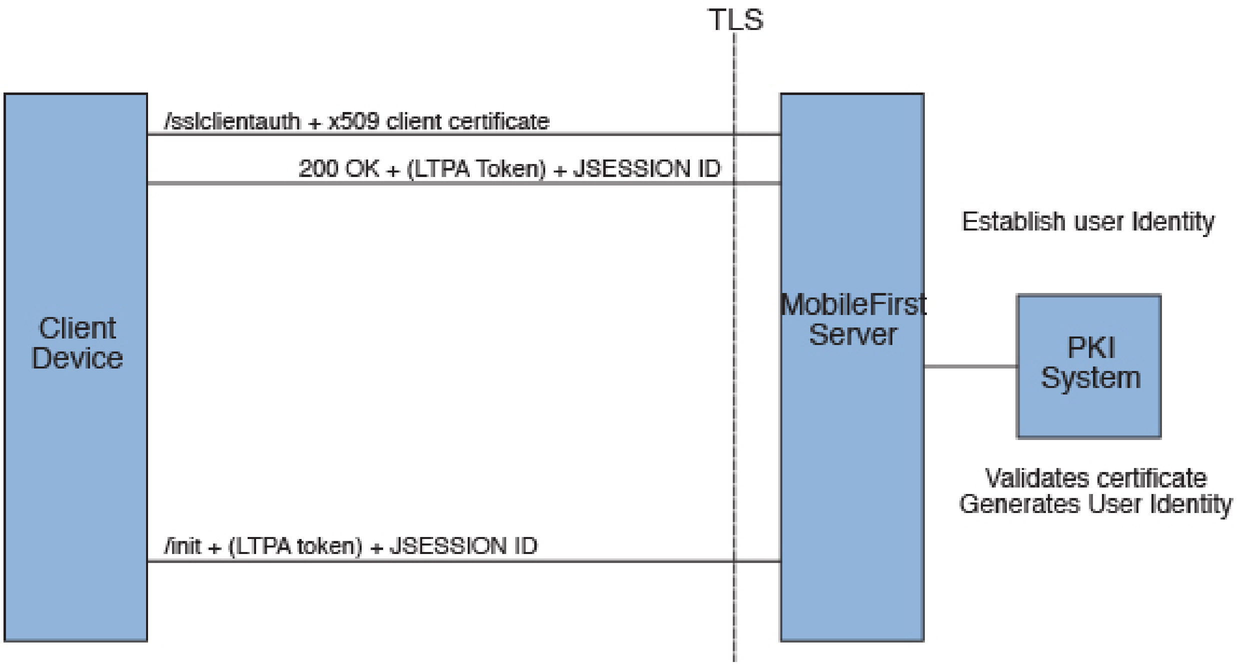


Subsequent calls from that MobileFirst application use this X.509 certificate to establish a secure connection over HTTPS, authenticate the user, and establish the user identity on the server. Users need to log in only once for the life of the certificate. When the certificate expires or is revoked by 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:



User certificate authentication on the server

Both the MobileFirst 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 MobileFirst 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 the Secure Sockets Layer (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 8-240.

The User Certificate Authentication feature requires integration with a public key infrastructure (PKI). For the embedded PKI option, you must 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.

If you encounter errors with certificates that are not signed by well-trusted CAs, see “Configuring SSL by using untrusted certificates” on page 6-180.

Restriction: 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 Client X.509 Certificate Authentication and

User Enrollment and the other Authentication and security tutorials on the Getting Started page of the Developer Center.

Certificate chains, keystore, and truststore

You must set the server certificate as the MobileFirst Server keystore. Also, set the client’s certificate-signing CA as part of the truststore 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 8-240.

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 MobileFirst 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 MobileFirst user identity

The standard MobileFirst 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 MobileFirst user identity
  - User name
  - Display name
  - Attributes
- Device ID
- Application name

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 MobileFirst custom certificate extensions to the user certificate before it is signed. This parameter provides more details to user identity attributes on subsequent runs. These details include device ID, group ID, and application name that is stored in the certificate. By default, this parameter is `false`. You can enable the parameter by using the `true` value. This parameter is not always supported and may not work for your configured server configuration. You must test this option first on your infrastructure to ensure that a certificate is not marked invalid if extensions are enabled. When this parameter is enabled, the device ID is added with the OID 1.3.6.1.4.1.2.6.256.1 and the app name is added with the OID 1.3.6.1.4.1.2.6.256.2. These OIDs are not formally registered and may change.

embedded-pki-bridge-days-before-expire
Optional

Configure the length of time the generated certificate is valid. This setting defaults to one year (365 days).

embedded-pki-bridge-crl-uri
Optional

Configure an optional CRL for your certificate authority. If the certificate that is generated exists on a client's device and is revoked in the CRL, the client is required to generate a certificate.

External/adapter-based PKI bridge:

The adapter-based PKI bridge is an included PKI bridge that can be used with user certificate authentication. The adapter-based PKI bridge is available with the com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI class name, and is configured by adding parameters to the realm definition. An adapter is required for this PKI bridge to work, and must be uploaded before any user connects with this configuration. The adapter-based PKI bridge is useful if your PKI can be accessed with an adapter (such as a REST API).

Requirements for use

For the adapter-based PKI bridge, an adapter must be added in the console and the parameters for the bridge must be configured in the realm definition. The following example shows a realm definition for wl_userCertificateAuthRealm with the adapter-based PKI that uses an adapter that is called PKIAdapter:

```
<realm name="wl_userCertificateAuthRealm"
       loginModule="UserCertificateLoginModule">
  <className>com.worklight.core.auth.ext.UserCertificateAuthenticator</className>
  <parameter name="dependent-user-auth-realm"
             value="WASLTPARealm" />
  <parameter name="pki-bridge-class"
             value="com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI" />
  <parameter name="adapter-pki-bridge-init-procedure"
             value="PKIAdapter.init" />
  <parameter name="adapter-pki-bridge-identity-validation-procedure"
             value="PKIAdapter.validateIdentity" />
  <parameter name="adapter-pki-bridge-csr-requirements-procedure"
             value="PKIAdapter.getCSRRequirements" />
  <parameter name="adapter-pki-bridge-csr-validation-procedure"
             value="PKIAdapter.validateCSR" />
  <parameter name="adapter-pki-bridge-certificate-generation-procedure"
             value="PKIAdapter.generateCertificate" />
  <parameter name="adapter-pki-bridge-identity-from-certificate-procedure"
             value="PKIAdapter.getIdentityFromCertificate" />
  <parameter name="adapter-pki-bridge-certificate-validation-procedure"
             value="PKIAdapter.validateCertificate" />
</realm>
```

Configuration parameters

The following adapter-based PKI bridge parameters are available.

adapter-pki-bridge-init-procedure
Required

An adapter procedure that is called to initialize the PKI bridge on each call. Requires a single parameter for the configuration that is available in the realm definition. The following example shows a sample value of this parameter:

```
{"adapter-pki-bridge-csr-validationprocedure":"
PKIBridgeAdapter.validateCSR","adapter-pki-bridge-identity-fromcertificate-
procedure":"PKIBridgeAdapter.identityFromCertificate","pkibridgeclass":"
com.worklight.core.auth.ext.UserCertificateAdapterBasedPKI","adapterpki-
bridge-identity-validationprocedure":"
PKIBridgeAdapter.identityVerify","adapter-pki-bridge-csrrequirements-
procedure":"PKIBridgeAdapter.csrRequirements","adapter-pkibridge-
certificate-generationprocedure":"
PKIBridgeAdapter.generateCertificate","adapter-pki-bridgecertificate-
validationprocedure":"
PKIBridgeAdapter.certificateVerify","adapter-pki-bridge-initprocedure":"
PKIBridgeAdapter.init","dependent-user-authrealm":"
WASLTPARealm"}
```

adapter-pki-bridge-identity-validation-procedure

Optional

An adapter procedure that is called that allows the adapter to determine whether the user identity from the dependent realm is allowed to generate a certificate. This procedure is optional. By default, the PKI bridge always returns `YES`. Requires a single `userIdentity` parameter. The following example shows a sample value of this parameter:

```
{"deviceId":"C146B473-DA25-46A7-8A79-E8CE5E9270EE","userIdentity":
{"userName":"user@ibm.com","attributes":
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

The procedure must return an object with the following format:

```
{valid: "YES"}
```

Options for `valid`:

- `YES` - The user is allowed to generate a certificate.
- `NO_USE_DEPENDENT_REALM_ONLY` - The user is allowed to log in to the dependent realm, but is not allowed to generate a certificate.
- `NO` - The user is not allowed to log in at all, and is not allowed to generate a certificate.

adapter-pki-bridge-csr-requirements-procedure

Optional

Build a set of requirements that must be in a CSR that the client generates. This procedure is optional. By default, the CSR requirements include the `commonName` that is equal to the user name from the dependent realm user identity. The procedure has a single parameter that is called `userIdentity` with the following format:

```
{"deviceId":"C146B473-DA25-46A7-8A79-E8CE5E9270EE","userIdentity":
{"userName":"user@ibm.com","attributes":
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

This procedure must return a JSON object in the following format:

```
{ commonName: "user@ibm.com", additionalSubject: { "O": "IBM" }, additionalAttributes: {} }
```

- commonName - This attribute is a required entry that is used as the `CN` attribute in the CSR. This value must match a user in the user registry of the application server.
- additionalSubject - This attribute is a required JSON object that contains key/value pairs for each additional attribute that must be in the subject of the CSR, such as `O` for organization. If no additional attributes are required, use an empty JSON object.
- additionalAttributes - This attribute is a required JSON object that contains key/value pairs for each additional attribute that must be included in the CSR. If no additional attributes are required, use an empty JSON object.

`adapter-pki-bridge-csr-validation-procedure`

Optional

This procedure is called after a client sends a CSR that follows a request. It is responsible for ensuring that all of the CSR attributes that were requested in the requirements exist in the CSR. This procedure is optional. By default, the PKI bridge always returns YES. The procedure has a single parameter `csr` that contains a JSON object with the following format:

```
{"csr":"MIICXzCCAUcCADAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjANBgkqhkiG9w0
BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZKcf3wW2LhQ75
MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2QggqpBMFPhvBdTbS
93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABoAAwDQYJKoZIhvcNAQEFBQADggEBAHHOJbrGBCZCiDi3hXzVzji7
1euKMf8IUjGe+sfr+Sy5sfx9k
+icvKixImHCxSy0PeKp4QICSgfZxk2xQzHhYVgdeB0Uv2WT7FjPngRjAgLL1jxu7LIkEMKWgiGiJMPg
54gOx8kWuj5uE9vqpWGRK0dGuPNlnQxh50pSgZi4PhRGz2nCBF6WdQFNmHDqssijk//
CUHWbNvMTIWyuHhXEhtwkplc0dAp1b3hHBywYM9Vae9fUmfpbHDb0yvjBjCHvceRjwkoQG6ABfh9
9ucE1NWO51Rc03XqGnHKsnk16BlqSH0YpM/sVWYrmio/F9h75aNX+Sz5EhkB7t/n4301aPOo="}
```

Note: `csr` is the CSR in DER format and is represented in `base64`.

The procedure must return a JSON object with the following format:

```
{valid: "YES"}
```

or

```
{valid: "NO"}
```

Options for `valid`:

- `YES` - The CSR meets the requirements from the PKI.
- `NO` - The CSR does not meet the requirements from the PKI. Authentication fails.

`adapter-pki-bridge-certificate-generation-procedure`

Required

This procedure is responsible for requesting a certificate from the PKI and returning a certificate. This procedure is required and has one required parameter `csr`, which has the following format:

```
{"deviceId":"C146B473-DA25-46A7-8A79-
E8CE5E9270EE","csr":"MIICXzCCAUcCADAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjA
NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZK
cf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2Qggq
pBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABoAAwDQYJKoZIhvcNAQEFBQADggEBAHHOJbrGBCZCiDi3hXzVzji7
1euKMf8IUjGe+sfr+Sy5sfx9k
+icvKixImHCxSy0PeKp4QICSgfZxk2xQzHhYVgdeB0Uv2WT7FjPngRjAgLL1jxu7LIkEMKWgiGiJMPg
54gOx8kWuj5uE9vqpWGRK0dGuPNlnQxh50pSgZi4PhRGz2nCBF6WdQFNmHDqssijk//
CUHWbNvMTIWyuHhXEhtwkplc0dAp1b3hHBywYM9Vae9fUmfpbHDb0yvjBjCHvceRjwkoQG6ABfh9
9ucE1NWO51Rc03XqGnHKsnk16BlqSH0YpM/sVWYrmio/F9h75aNX+Sz5EhkB7t/
n4301aPOo=","userIdentity":{"userName":"lizet@us.ibm.com","attributes":
{"LtpaToken":"dHwRqHp61ukJCkEFBMRd6g63uV1bDg0rmGBU2cuBrinFp+7L7BVb
+4OebyIRMOoKLhHldLxj9JIPiWH4s16tHtNjddBxxbd9rdjZUgnicVY8+6GM8uTEwleRW
+lVzzwJX0Htvfa2iOQD9KAWLXkNHgneiELIANjAUxGsMzJGGg2K8LYYWeBhE0JGqJcb8WFFLYH4T5
Cgb9C+qXpre/KF/MNTrv2WQF9kWjPmMlPuT1Lh1tY9oSSqN20DNNZ8VcQ8p26po5yBMvtDMtn4/
EzfdhKYeTNFzQEmQpR66caQJlRV++m/
Oq4EiZBBzkOY6zpBVtmUzcH3D2xh5PYYVcFO8g=="},"displayName":""},"appId":"UserCert"}
```

Note: `csr` is the CSR in DER format and is represented in `base64`.

The procedure must return a `base64` string of the X.509 certificate in DER format in a JSON object with the following format:

```
{ certificateBase64: "<BASE64 STRING OF THE X.509 CERTIFICATE>" }
```

`adapter-pki-bridge-certificate-validation-procedure`

Optional

This procedure is responsible for validating a user's certificate when it is first received. This procedure is optional. If it is not used, the PKI bridge always returns YES. The procedure has one parameter `certificate` that is in the same format as the procedure in the `adapter-pki-bridge-identity-from-certificate-procedure` parameter.

The procedure is required to return a JSON object that states the validity of the certificate:

```
{valid: "YES"}
```

or

```
{valid: "NO"}
```

Options for `valid`:

- `YES` - The certificate is considered valid by the PKI.
- `NO` - The certificate is not considered valid by the PKI, and the client is required to start the enrollment process over.

`adapter-pki-bridge-identity-from-certificate-procedure`

Required

This procedure is responsible for creating a user certificate identity from a certificate that is passed by the user. The procedure must have one parameter `certificate` with the following format:

```
{"publicKey":
{"base64":"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLC
CGEBCUQAgNPZKcf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv8
1+wHaTIu2QggqpBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQAB","algorithm":"RSA"},"signature":
{"base64":"cONA8EKOQBiIKtdhAzG68pm0FMRkNfbVAIyZlttp+J9nXYmjO/
aGOEJk37oGzEPTO5uA/
eDArvQ9WF3BtzOdF9hw4j3ACJjo5oEnD7UTXbPzK2k1w3INX4cuOInLi7EJEKb
+CuO5uMy1mUOjx1aj/WaK
+E2KroFKNPyXdHAL7mwpkZO0aSYxUYYwcu8IAureMWZGps196Swk1YptboIEUSd5r3j07rBZX81B
AX5awqEx3tpbP3qpIJIK+6xoiu2tL67mKqJj9l1/Yb/
qQmUg6ouJtt9fWYUO7p1wJgUm9N0eixXftKttJ32Fp/
s0B7R72nt09pGPrkYt8IUkzSq22Q==","algorithm":"SHA1withRSA"},"subjectUniqueId":"","version"
:1,"issuer":{"dn":"CN=Worklight Test Beta Signing CA,OU=Security Division,O=IBM
Worklight,L=Austin,ST=TX,C=US","cn":"Worklight Test Beta Signing
CA","uniqueId":""},"dn":"CN=user@us.ibm.com","cn":"user@us.ibm.com","valid":{"notBefore":
1381193593,"notAfter":
1382403193},"serialNumber":"efa7b0e3f0d9cef0","base64":"MIIDIzCCAgsCCQDvp7Dj8NnO8DA
NBgkqhkiG9w0BAQUFADCBizELMAkGA1UEBhMCVVMxCzAJBgNVBAgTAlRYMQ8wDQYDVQQHEwZ
BdXN0aW4xFjAUBgNVBAoTDUlCTSBXb3JrbGlnaHQxGjAYBgNVBAsTEVNlY3VyaXR5IERpdmlzaW9u
MSowKAYDVQQDEyFXb3JrbGlnaHQgR2FycmljayBCZXRhIFNpZ25pbmcgQ0EwHhcNMTMxMDA4M
DA1MzEzWhcNMTMxMDIyMDA1MzEzWjAbMRkwFwYDVQQDFBBsaXpldEB1cy5pYm0uY29tMIIBIjA
NBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAt9A8WLa0NqAjizn9ccZFUPBLCCGEBCUQAgNPZK
cf3wW2LhQ75MEMfLyahZvqSBFd7IMMstRrpKiobx6PTGiMCkNB7lOzNa88tCHv81+wHaTIu2Qggq
pBMFPhvBdTbS93pafEQ7kXEGBk+uU7vwalUIHQyQT1+9ZaiH4ssf8Ybi
+qYmGrOH4CjvO7h93l0sAyOOWqcGBnOCcb1+YJP9F/
EyHLNfdr1FTDAAp0ERtUqVMDeJIRxscFnqZ1GG0rXCEJqAl3IHvrn6BiLrmQOxA5oE
+Lk4ry6cizw1yxYY1mWZq9eTCQQbMGBS/Aa+4KBOG3NCCL
+e4YKN2RJ0m2bcHRswIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQBw40DwQo5AGIgq12EDMbry
mbQUxGQ19tUAjJmW22n4n2ddiaM79oY4QmTfugbMQ9M7m4D94MCu9D1YXcG3M50X2HDiPcA
ImOjmgScPtRNds/MraTXDcg1fhy44icuLsQkQpv4K47m4zLWZQ6PHVqP9Zor4TYqugUo0/
Jd0cAvubCmRk7RpJjFRhjBy7wgC6t4xZkamzX3pLCTVim1uggRRJ3mvePTusFlfzUEBflrCoTHe2ls/
eqkgkgr7rGiK7a0vruYqomP2XX9hv+pCZSDqi4m2319ZhQ7unXAmBSb03R6LFd+0q20nfYWn
+zQHtHvae072kY+uRi3whSTNKrbZ"}
```

Note: `base64` is the DER formatted certificate. `publicKey` is also encoded in `base64`.

The procedure must return a JSON object in the following format:

```
{ userName:"user@us.ibm.com",displayName:"",attributes:{},appID:"UserCert",deviceId:"C146B473-DA25-46A7-8A79-E8CE5E9270EE" }
```

The goal of the JSON object that is returned is to form the original user identity of the user that is provided by the dependent realm during generation.

Note: `appId` and `deviceId` are optional in this step. If not used, use an empty string as the value.

Custom PKI bridge:

A custom PKI bridge can be implemented by extending the `com.org.auth.ext.UserCertificatePKIBridge` abstract class.

The API for the PKI bridge abstract class can be found at UserCertificatePKIBridge.

WebSphere Application Server and Liberty profile requirements

User certificate authentication uses standard SSL X.509 User Certificates, which requires the use of an SSL channel.

There are a few requirements around SSL that must be configured in order for user certificate authentication to work.

- The SSL channel for WebSphere Application Server or the Liberty profile must be configured to include the certificate authority (CA) in the trust store that is used to sign user certificates.
- The application server must be configured to allow a user certificate, but not require it. This configuration is important so that IBM MobileFirst Platform Foundation for iOS 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 MobileFirst client cannot handle this configuration. If you want to protect the MobileFirst 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 MobileFirst 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 MobileFirst Operations 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 8-234) or adapter-based ("External/adapter-based PKI bridge" on page 8-236) or supply your own custom PKI bridge implementation ("Custom PKI bridge" on page 8-240).
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 8-234) or Adapter-Based ("External/adapter-based PKI bridge" on page 8-236) 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 "Security tests" on page 8-223. 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 MobileFirst 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 6-180.

Dependent user realm

The first time a user attempts to connect to the server, IBM MobileFirst Platform Foundation for iOS tries to enroll the user into the PKI and provision the device with the user certificate. To enroll the user, IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 8-244.

Clearing certificates on the chain

Certificates on the client are managed by the MobileFirst 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. 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 MobileFirst 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 MobileFirst family.

About this task

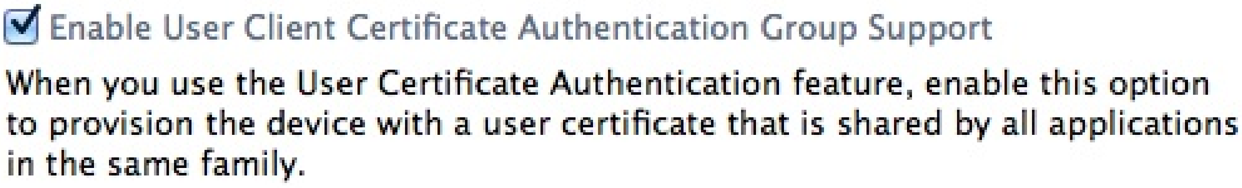
You can configure the User Certificate Authentication feature to provision the device with a user certificate that is shared among a group of applications. This configuration allows a group of applications to authenticate with the same X.509 client certificate.

Note: The iOS `x509AccessGroup` property is deprecated since IBM MobileFirst Platform Foundation for iOS V7.1.0. Use the Simple Data Sharing feature instead.

Procedure

1. Enable the Simple Data Sharing feature.
   a. For iOS applications, follow the steps in "Enabling the Simple Data Sharing feature for iOS native applications" on page 8-247.

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 8-17. 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 MobileFirst Server on your device. For example, go to the MobileFirst Operations 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 8-243.
`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 8-233. 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 MobileFirst 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 8-17. 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.

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 MobileFirst 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 applications.

After you enable the Simple Data Sharing feature, you can use the provided native APIs to exchange simple string tokens among a family of applications on a device.

When used with the MobileFirst device single sign-on (SSO) feature, 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.

For more information about device SSO with a reverse proxy, see "Configuring device single sign-on with a reverse proxy" on page 8-296.

Simple data sharing general terminology

Learn about simple data sharing general terminology.

MobileFirst 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 MobileFirst 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.
- 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.

Aside from the IBM MobileFirst Platform Foundation for iOS provided APIs, applications in the same MobileFirst 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 MobileFirst application family can now be accomplished in native iOS 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 8-149.

Enabling the Simple Data Sharing feature

Learn how to enable the Simple Data Sharing feature.

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" on page 8-7.

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.
2. In Xcode, add a Keychain Access Group with the same name as your `wlAppFamily`.

   The application-identifier entitlement must be the same for all applications in your family.

   Note: By default, MobileFirst 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.

Simple data sharing API concepts

Learn about simple data sharing API concepts.

Sharing string tokens across applications of the same MobileFirst application family can be accomplished in iOS 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.

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.

Troubleshooting simple data sharing

Find information to help resolve issues that you might encounter when you use the Simple Data Sharing feature.

Table 8-18. 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 MobileFirst application family name. For more information, see “Enabling the Simple Data Sharing feature” on page 8-247.

Table 8-18. Troubleshooting the Simple Data Sharing feature (continued). This table lists possible problems and actions to take to troubleshoot the Simple Data Sharing feature.

Problem	Actions to take
Unable to get MobileFirst 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 8-247. 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 8-296.
Unable to specify cookie or user certificate sharing.	You must first enable the MobileFirst Simple Data Sharing feature and specify a MobileFirst 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 8-247.

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 8-149.

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.

Mobile device authentication

You can require mobile devices to authenticate themselves. Device identity is used in several places within IBM MobileFirst Platform Foundation for iOS. 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 MobileFirst Platform Foundation for iOS 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.

Note: The availability of the unique device ID depends on the operating system of the device, and on the application vendor. A vendor who 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 MobileFirst 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. This certificate results of the provisioning process that takes place the first time the application runs on the device.

The unique device ID is sent along with various event types to MobileFirst Operational Analytics (see “Operational analytics” on page 13-6). For more information on these event types, see “Event types” on page 13-10.

The unique device ID is also stored in the raw data reports (deprecated) that are generated by IBM MobileFirst Platform Foundation for iOS. 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 13-98.

For more information about mobile device provisioning, see the Device provisioning conceptss tutorial on the Getting Started page of the Developer Center.

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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst Server automatically issues a certificate for the device and application data that is provided by the client application. Use this option only when the MobileFirst application authenticity features are enabled.

Custom provisioning
In this mode, the MobileFirst 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 MobileFirst Server to do so.

Note:

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 MobileFirst 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 MobileFirst Server. The MobileFirst 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 MobileFirst 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.

Authenticators and login modules

An authenticator collects client credentials. A login module validates them.

An *authenticator* is a server component that is used to collect credentials from the client. The authenticator passes the credentials to a *login module*, which validates them and builds a client identity object. Both authenticators and login modules are components of the application's *realm*.

An authenticator can, for example, collect any type of information accessible from an HTTP request object, such as cookies or any data in headers or the body of the request.

A login module can validate the credentials that are passed to it in various ways. For example:

- Using a web service
- Looking up the client ID in a database
- Using an LTPA token

A number of predefined authenticators and login modules are supplied. If these do not meet your needs, you can write your own in Java.

The authentication configuration file

All types of authentication component are configured in the authentication configuration file.

Authentication components, security tests, realms, login modules, and authenticators are all configured in the `authenticationConfig.xml` authentication configuration file, which is in the `/server/conf` directory of your MobileFirst 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 8-19. 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 MobileFirst Platform Foundation for iOS 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 back-end systems, and push notification. This realm is transparent, that is, it does not require any user interaction. The **wl_anonymousUserRealm** realm is a persistent cookie and is used by MobileFirst Server to differentiate between different application instances. MobileFirst Server looks for a **WL_PERSISTENT_COOKIE** cookie in each request. In case none is found, it will generate a GUID and send it back to the client in a Set-Cookie header with a one-year expiration date.
wl_antiXSRFRealm	**WLAntiXSRFLoginModule**	This realm is used to avoid cross-site request forgery attacks. When a new session is initiated, the first request to MobileFirst Server gets an HTTP 401 response that contains the WL-Instance-Id token. The MobileFirst 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.

Table 8-19. Predefined realms: properties of the **<test realm>** *element. (continued)*

Realm reference	Login module reference	Description
wl_authenticityRealm	**wl_authenticityLoginModule**	This realm is used to verify that application is authentic and it was not modified by a third party. The realm is applicable to Android, iOS, Windows Phone Silverlight 8, and Windows 8 Universal. The basic authenticity check is based on certificates that are used to sign applications. Extended application authenticity is based on the application binary file. This functionality is only available on the IBM-supported editions of IBM MobileFirst Platform Foundation for iOS, and is supported by Android, iOS, Windows 8 Universal, and Windows Phone Silverlight 8 only. You cannot enable extended authenticity if you use the internal MobileFirst Development Server that is embedded in MobileFirst Studio.
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 MobileFirst Server. This realm must be used with **wl_authenticityRealm**.
wl_deviceNoProvisioningRealm	**WLDeviceNoProvisioningLoginModule**	A default *device identity* realm. Device identity is similar to user identity, but it is provided by the device itself. Device identity is relevant for hybrid and native smartphone environments only. The device identity is a must for functionality such as push notifications, and reports. This parameter means that the obtained device identity is used as is, without provisioning.

Table 8-19. Predefined realms: properties of the **<test realm>** element. (continued)

Realm reference	Login module reference	Description
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..
wl_remoteDisableRealm	**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 12-3.

MobileFirst static resources (other than Application Center) such as the MobileFirst Operations Console are also configured in the authentication configuration file, in the **<resource>** element.

The configuration file has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<tns:loginConfiguration xmlns:tns="http://www.worklight.com/auth/config" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <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 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: All applications that are part of the same group 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 MobileFirst CA certificate, configure the following properties.

  wl.ca.keystore.path

  The path to the keystore, relative to the server folder in the MobileFirst 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 MobileFirst configuration properties, see "Configuration of MobileFirst applications on the server" on page 11-50

- To enable multiple applications to share the same certificate, define a **`bundleId`** attribute in the application descriptor.

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. Specify the names of the `validate-csr` and `validate-certificate` functions 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.validateCertificate"/>
  </loginModule>
</loginModules>
```

The *`validate-csr-function`* checks 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.

The *`validate-certificate-function`* verifies that the certificate was issued with the correct certificate authority (CA). The logic might also verify that the certificate contains all the necessary data about the device for this custom device authentication realm.

For more information about how to implement these functions, see the Custom device provisioning tutorial on the Getting Started page.

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 8-250.

With custom device provisioning, you can also implement custom variations of the CSR during the initial provisioning flow and of the certificate at each application start.

You must configure the server and the client for custom device provisioning.

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 and name it `ProvisioningAdapter`.
2. Add two functions with the following signatures to the adapter's JavaScript file:
   - The `validateCSR(clientDN, csrContent)` function is called 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 called again.
   - The `validateCertificate(certificate, customAttributes)` function is called each time that the mobile application establishes a new session with the MobileFirst 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 MobileFirst Server.

   Note: These functions are called internally by the MobileFirst authentication framework. Do not declare them in the adapter's XML file.
3. Configure the `authenticationConfig.xml` file.
   a. Add a realm and name it `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 validates the certificate signing request (CSR).

      ```
      <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 validates the certificate.

```
<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

`validateCSR` function

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;
}
```

`validateCertificate` function

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 tutorial on the Getting Started page.

Implementing client-side components for custom device provisioning:

You can implement client-side components for custom device provisioning.

The following prerequisites are required for device provisioning:
- MobileFirst Server
- In the Application Center console, application authentication must be set to `enabled, blocking`.

The included MobileFirst Development Server can be used for device provisioning.

The following sections describe the implementation of the client-side components in iOS applications.

Implementing client-side components for native iOS:

You can implement client-side components for custom device provisioning in native iOS.

Before you begin

For more information about the prerequisites, see "Implementing client-side components for custom device provisioning."

About this task

To implement client-side components for custom device provisioning, complete the following steps.

Procedure

1. Create a MobileFirst native API application for iOS.
2. Configure the application for the Application Authenticity test.

   Note: The authenticity test does not work if you use the internal MobileFirst Development Server that is embedded in MobileFirst Studio. For more information about application authenticity, see "MobileFirst application authenticity overview" on page 8-221.
3. Create an iOS native application and use the `wlConnectWithDelegate` function to connect to the server.
4. For the `wlConnectWithDelegate` function to trigger authentication, specify the MobileFirst native API application as a protected resource by adding a custom security test or mobile security test in the application descriptor.

   ```
   <nativeIOSApp securityTest="MySecurityTest" version="1.0">
   ```
5. Add a new class `CustomChallengeHandler`and register it in the main by using `[[WLClient sharedInstance] registerChallengeHandler:[customChallengeHandler initWithRealm:@"wl_myCustomProvisioningRealm"]]`.

6. Implement the following methods, which are required by the challenge handler for device provisioning.

   `createCustomCsr(challenge)`
   This method is responsible for returning custom properties that are added to the certificate signing request (CSR). Add a custom **`activationCode`** property, which is used in the adapterâ€™s `validateCSR` function.

   `handleSuccess(identity)`
   This method is called when certificate validation is successfully completed by the `validateCertificate` adapter function.

   `handleFailure()`
   This method is called when certificate validation fails. You must call `clearDeviceProvisioningCertificate()` from this method to delete the stored certificate on the device.

   Here is a sample implementation of a challenge handler for custom device provisioning:

```
@interface CustomChallengeHandler : BaseProvisioningChallengeHandler <WLDelegate>{
@private
    ViewController *vc;
}
- (id)initWithController: (ViewController *)mainView;
- (void) createCustomCsr : (NSDictionary *) challenge;
@property (nonatomic, strong)NSString *passcode;
@end


@implementation CustomChallengeHandler
- (id)initWithController: (ViewController *) mainView{
    if ( self = [super init] )
    {
        vc = mainView;
    }
    return self;
}
-(void) createCustomCsr : (NSDictionary *) challenge {
    [vc updateMessage:@"\nCreating custom Csrâ€⌂];
    [vc updateMessage:[NSString stringWithFormat:@"\t Passcode :: %@", self.passcode]


    NSMutableDictionary* answer =[[NSMutableDictionary alloc] init];
    [answer setValue:self.passcode forKey:@"activationCode"];
    [self submitCsr:answer :challenge];
}
-(void)onSuccess:(WLResponse *)response {
    [vc updateMessage:@"Device authentication with custom device provisioning was successfully completed"];
    [vc updateMessage:response.description];
}
-(void)onFailure:(WLFailResponse *)response{
    [vc updateMessage:@"Server has rejected your device. You must reinstall the application and perform
device provisioning again."];
    [vc updateMessage:response.description];
}
@end
```

Results

You have implemented client-side components for custom device provisioning in native iOS.

What to do next

You can implement server-side components for custom device provisioning. For more information, see “Implementing server-side components for custom device provisioning” on page 8-259. For more information about custom device provisioning, see the tutorial on the Getting Started page.

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
`expirationInSeconds`	Optional. Defines the expiration period of realms that use this login module. If not set, the server will use a default expiration period of 3600 seconds (one hour). Note that if the server is not running in session-independent mode, authentication to a login module is only valid within the same client session. In session-dependent mode only, you can use the special value "-1" that indicates that there is no set expiration period and the login remains valid until the end of the client session. For more information about session-independent mode, see “Session-independent mode” on page 8-108.
`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 MobileFirst security framework provides several built-in realms (for example, directUpdate and remoteDisable). In order to modify the default expiration period of these realms, open the `worklight.properties` file, uncomment the line corresponding to the realm, and change the expiration value. The following shows the relevant sections in `worklight.properties`:

```
########################################################################################################################
#    Expiration time for built-in realms
########################################################################################################################
# Use these properties to configure the expiration time (in seconds) for MobileFirst's built-in realms.
# When mfp.session.independent is false, a value of -1 means that a realm will remain authenticated
# until the session times out
wl.realm.expiration.directUpdate=3600
wl.realm.expiration.remoteDisable=300
wl.realm.expiration.deviceAutoProvisioning=3600
wl.realm.expiration.deviceNoProvisioning=3600
wl.realm.expiration.antiXSRF=3600
wl.realm.expiration.authenticity=3600
wl.realm.expiration.anonymousUser=3600
```

Note: To avoid application-authenticity validation failures, use the same expiration periods for the application-authenticity and device-provisioning realms. The default expiration periods are identical, but if you customize the default period of the authenticity, deviceAutoProvisioning, or deviceNoProvisioning realms, set the same period also for the other two realms.

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.

Important: Changes to realm configuration and login module configuration (excluding changes to the expiration period) are not applied immediately to clients that authenticated within the realm before the change. The changes will be applied only when the realm expires or when the client has logged out of the realm explicitly. Consider this behavior if you plan to use a long expiration period: in addition to the security implications, a long expiration period also limits the flexibility of making changes to the realm configuration.

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 11-61.

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 IBM MobileFirst Platform Foundation for iOS is run on other application servers, the login module is commented out in the default `authenticationConfig.xml` file that is created with an empty MobileFirst project. To use it, remove the comments first.

Class Name

`com.worklight.core.auth.ext.WebSphereLoginModule`

Parameters

The login module class has the following parameters:

Parameter	Description
`cookie-domain`	Optional. A String such as `example.com`, which specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The single sign-on is then restricted to the application server host name and does not work with other hosts in the same domain.
`httponly-cookie`	Optional. A String with a value of either `true` or `false`, which specifies whether the cookie has the HttpOnly attribute set. This attribute helps to prevent cross-site scripting attacks.
`cookie-name`	Optional. A String that specifies the name of the LTPA SSO cookie. If this parameter is not set, the default cookie name is `LtpaToken`.
`role`	Optional. A String that specifies the Java EE role that the authenticated user must belong to for the login to be successful. If the parameter is not specified, no role checking is performed.

Note: When you specify a role parameter, the role must be defined in the MobileFirst 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 lightweight directory access protocol (LDAP) servers such as Active Directory or OpenLDAP.

The class of the LDAP login module is LdapLoginModule.

Note: The LDAP login module implements a UserNamePasswordLoginModule interface. Therefore, you must use this module with an authenticator that implements a UsernamePasswordAuthenticator interface.

Class Name

`com.worklight.core.auth.ext.LdapLoginModule`

Validation methods

The value of the **validationType** configuration parameter of the LDAP login module determines the method that is used to validate the credentials that are received from the client, and authorize the client:

exists This method checks whether the provided client credentials exist in the LDAP server: the login module attempts to bind to the LDAP server with the provided credentials. If the binding is successful, the credentials validation succeeds. Otherwise, the validation fails.

searchPattern

This method performs a specified search in the LDAP server, as a condition for the credentials validation: the login module first attempts to bind to the LDAP server with the provided client credentials, as done in the `exists` validation method. If the binding succeeds, the login module searches the LDAP server using the search criteria that are set in the **ldapSearchFilterPattern** and **ldapSearchBase** parameters. If the search returns one or more entries, the credentials validation succeeds. Otherwise, the validation fails.

The following LdapLoginModule configuration parameters are mandatory when using the `searchPattern` validation method:

- **ldapSearchFilterPattern** - The search filter to use in the LDAP search. Use `{username}` in the pattern as a placeholder for the client-credentials user name.

  Example

  ```
  (&amp;(objectClass=user)(sAMAccountName={username})(memberof=CN=goodgroup,OU=Groups,OU=
  ```

- **ldapSearchBase** - The search base for the LDAP search.

  Example

  ```
  dc=myserver,dc=com
  ```

searchUniqueUser

This method consists of two parts: search the LDAP server using the credentials of a system user; then validate the client credentials by binding to the server with a unique user name that is derived from the search result. The login module performs the following steps; each step depends on the success of the previous step:

1. Bind to the LDAP server with the system-user credentials that are provided in the **ldapSearchUserName** and **ldapSearchPassword** parameters.
2. Perform an LDAP search using the search criteria that are provided in the **ldapSearchFilterPattern** and **ldapSearchBase** parameters.
3. Verify that the LDAP search returns a single user entry. If the search returns no match or more than one match, the validation fails.
4. Bind to the LDAP server with the password that is provided in the client credentials, and the user name that is derived from the search result according to the provided parameters:
   - When the value of the **ldapUseAttributeToExtract** parameter is `false`, the user name is the name of the user entry that is returned by the LDAP search.
   - When the value of the **ldapUseAttributeToExtract** parameter is `true`, the user name is the value of the attribute that is defined in the **ldapAttributeToExtract** parameter, which the login module extracts from the user entry that is returned by the LDAP search.

5. Verify that the binding succeeds.

The following LdapLoginModule configuration parameters are mandatory when using the `searchUniqueUser` validation method:

- **`ldapSearchUserName`** - The user name that is used for searching the LDAP server.

  Example

  ```
  dc=worklight,dc=lan
  ```

- **`ldapSearchPassword`** - The password that is used for searching the LDAP server.

  Example

  ```
  abcd!234
  ```

- **`ldapSearchFilterPattern`** - The search filter to use in the LDAP search. Use `{username}` in the pattern as a placeholder for the client-credentials user name.

  Example

  ```
  (&amp;(objectClass=user)(sAMAccountName={username})(memberof=CN=goodgroup,OU=Groups,OU=Wor
  ```

- **`ldapSearchBase`** - The search base for the LDAP search.

  Example

  ```
  dc=myserver,dc=com
  ```

- **`ldapUseAttributeToExtract`** - Indicates whether to use the **`ldapAttributeToExtract`** parameter. This parameter determines the user name that the login module uses to bind to the LDAP server:
  - `false` - Do not use the **`ldapAttributeToExtract`** parameter. When set, the login module binds to the server using the name of the user entry that is returned by the LDAP search.
  - `true` - Use the **`ldapAttributeToExtract`** parameter. When set, the login module binds to the server using the value of the **`ldapAttributeToExtract`** attribute, which it extracts from the user entry that is returned by the LDAP search.

The following LdapLoginModule configuration parameter is mandatory when the **`ldapUseAttributeToExtract`** parameter is set to `true`:

- **`ldapAttributeToExtract`** - The name of the attribute whose value should be extracted from the user entry that is returned by the LDAP search. The login module uses the extracted attribute value as the user name for binding to the LDAP server.

  Example

  ```
  distinguishedname
  ```

`custom` This method executes custom credentials-validation logic: the login module first attempts to bind to the LDAP server with the provided client credentials, as done in the `exist` validation method. If successful, the login module calls the doCustomValidation method of the doCustomValidationLdapLoginModule class, which performs custom validation.

Parameters

The LDAP login module has the following configuration parameters:

Parameter	Description	Sample values
ldapProviderUrl	Mandatory. The URL or IP address of the LDAP server.	ldap://10.0.1.2 ldaps://10.0.1.3
ldapTimeoutMs	Mandatory. The connection timeout for processing LDAP-server requests, in milliseconds.	2000
ldapSecurityAuthentication	Mandatory. The authentication method that is used by the LDAP server. Contact your LDAP administrator to obtain the correct value for your LDAP server. In most cases, the value is simple.	none simple strong
validationType	Mandatory. The method that is used by the login module to validate the credentials that are received from the client, and authorize the client. This parameter can be set to one of the following values; see "Validation methods" on page 8-267: • exists - Verifies that the provided client credentials exist in the server. See the exists validation method. • searchPattern - Searches the LDAP server using the specified search filter. See the searchPattern validation method. • searchUniqueUser - Searches the LDAP server with the specified system-user credentials, and binds to the server with a unique user name. See the searchUniqueUser validation method. • custom - Executes custom credentials-validation logic. See the custom validation method.	exists searchPattern searchUniqueUser custom

ldapSecurityPrincipalPattern	Mandatory. A pattern that defines the LDAP security principal to send to the LDAP server. The pattern depends on the syntax that is required by the LDAP server. Use username in the pattern as a placeholder for the user name that is provided in the client credentials.	{username}
		{username}@myserver.com
		CN={username},DC=myserver,DC=com

ldapSearchFilterPattern	Mandatory for the searchPattern and searchUniqueUser validation methods, and not applicable to the other validation methods. See the documentation of the applicable validation methods.	(sAMAccountName={username}) (&amp;(objectClass=user)(sAMAccountNam OU=MyCompany,DC=myserver,DC=com))
ldapSearchBase	Mandatory for the searchPattern and searchUniqueUser validation methods, and not applicable to the other validation methods. See the documentation of the applicable validation methods.	dc=myserver,dc=com

Parameter	Description	Sample values
ldapSearchUserName	Mandatory for the searchUniqueUser validation method, and not applicable to the other validation methods. See the documentation of the searchUniqueUser method.	@code dc=worklight,dc=lan
ldapSearchPassword	Mandatory for the searchUniqueUser validation method, and not applicable to the other validation methods. See the documentation of the searchUniqueUser method.	abcd!234
ldapUseAttributeToExtract	Mandatory for the searchUniqueUser validation method, and not applicable to the other validation methods. See the documentation of the searchUniqueUser method.	false
ldapAttributeToExtract	Mandatory for the searchUniqueUser validation method when **ldapUseAttributeToExtract** is true; not applicable to the other validation methods or when **ldapUseAttributeToExtract** is false. See the documentation of the searchUniqueUser method.	distinguishedname
ldapReferral	Optional. The referral-handling method to be used by the LDAP server. This parameter can be set to one of the following values: • ignore - Ignore referrals. • follow - Automatically follow referrals. • throw - Throw a referral exception (ReferralException) for every referral. When no value is set for this parameter, the login module does not instruct the LDAP server on how to handle referrals.	ignore

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"/>
  <parameter name="ldapReferral" value="ignore"/>
</loginModule>
```

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 the name of your extension class as the <class> name in <loginModule> elements in your AuthenticationConfig.xml file. The following sample definition uses the custom mycode.MyCustomLdapLoginModule class:

```
<loginModule name="LDAPLoginModule">
  <className>myCode.MyCustomLdapLoginModule</className>
  <parameter name="ldapProviderUrl" value="ldap://10.0.1.2"/>
  <parameter name="ldapTimeoutMs" value="2000"/>
  <parameter name="ldapSecurityAuthentication" value="simple"/>
  <parameter name="validationType" value="custom"/>
  <parameter name="ldapSecurityPrincipalPattern" value="{username}@myserver.com"/>
  <parameter name="ldapReferral" value="ignore"/>
</loginModule>
```

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 8-20. 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 8-21. 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.

Table 8-21. The `<realm>` element subelements (continued)

Element	Description
`<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, MobileFirst Server sends an HTTP Not Authorized (401) response to the client, with the header: `WWW-Authenticate: Basic realm="realmName"`. When MobileFirst 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 8-36. Basic authentication processBasic authentication process

Procedure

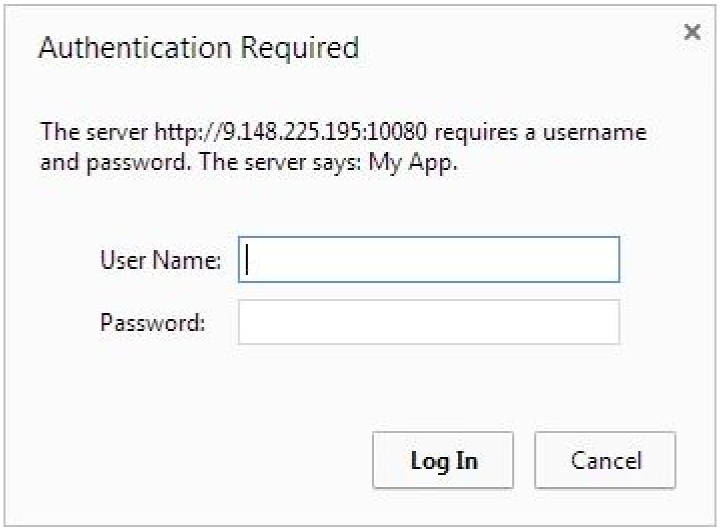
1. Configure the `authenticationConfig.xml` file. For more information, see "The authentication configuration file" on page 8-253.
2. Code the server side.

   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. MobileFirst Platform Command Line Interface for iOS 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.



Authentication Required

The server http://9.148.225.195:10080 requires a username and password. The server says: My App.

Figure 8-37. Login dialog for authentication

Configuration of the `authenticationConfig.xml` file

```
<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 MobileFirst 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:

   ```
   <procedure name="getSecretData" securityTest="DummyAdapter-securityTest"/>
   ```

Coding the client side

1. Create a MobileFirst 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 MobileFirst project. 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 MobileFirst Platform Foundation for iOS redirects to the page when the user credentials are needed.

Both the **login-page** and **auth-redirect** parameters 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 MobileFirst Platform Foundation for iOS uses its default login page template.

Flow

The following diagram illustrates the flow in the form-based authentication process:

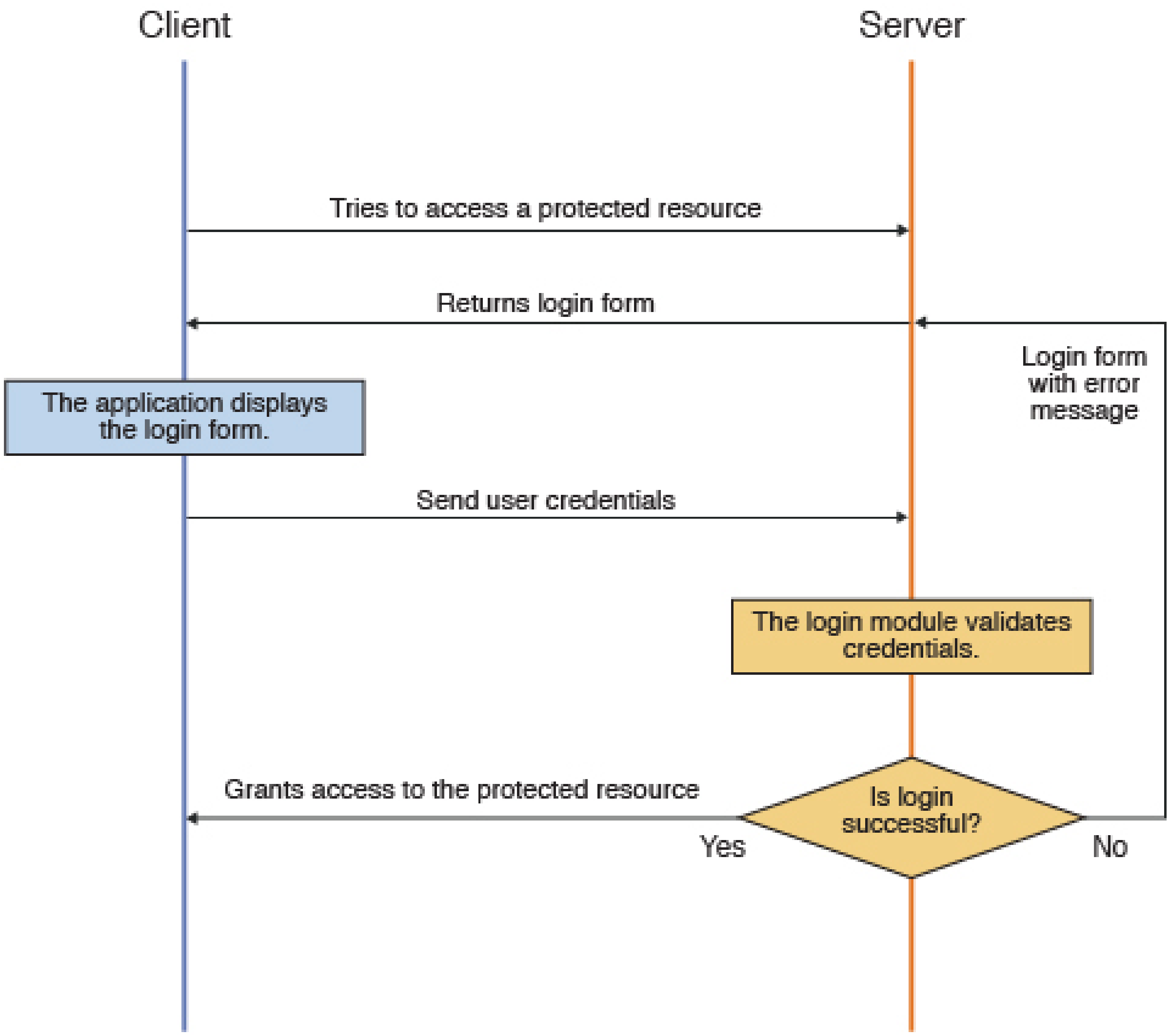

Client
Server
Tries to access a protected resource
Returns login form
Login form with error message
The application displays the login form.
Send user credentials
The login module validates credentials.
Grants access to the protected resource
Is login successful?
Yes
No

Figure 8-38. Form-based authentication process

Procedure

1. Configure the `authenticationConfig.xml` file. For more information, see "The authentication configuration file" on page 8-253.
2. Code the server side. To work, the form-based authenticator must be associated with a login module. MobileFirst Platform Command Line Interface for iOS 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 MobileFirst framework that the authentication finished successfully. The MobileFirst framework then automatically issues the original request that triggered the authentication.
- submitFailure: this function notifies the MobileFirst framework that the authentication process failed to finish. The MobileFirst 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 MobileFirst 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 MobileFirst 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 Form-based authentication tutorials on the Developer Center website.

Implementing Java-based custom authenticators

You can use default MobileFirst login modules and authenticators, or customize your own.

About this task

You can write custom login modules and authenticators when those that IBM MobileFirst Platform Foundation for iOS supplies do not match your requirements.

Procedure

1. Configure the `authenticationConfig.xml` file.

   For more information, see "The authentication configuration file" on page 8-253.
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.

   Important: To correctly support session-independent mode, observe the following guidelines. For a detailed overview of session-independent mode, refer to "Session-independent mode" on page 8-108.
   - Ensure that you do not save any applicative data in the HTTP session.
   - In your custom authenticator implementation, do not send ("flush") the prepared server-authentication response to the client. Sending the response while running in session-independent mode might lead to incorrect behavior if the client receives the response before the authentication context is saved to the attribute store. If the authenticator does not send the response to the client, MobileFirst Server does so automatically after all authentication-context information is saved to the attribute store.

   Authenticator interface and methods

   Your custom authenticator class must implement the `com.worklight.server.auth.api.WorkLightAuthenticator` interface. The `WorkLightAuthenticator` interface extends the `java.io.Serializable` interface, so as a result, your custom authenticator implements java.io.Serializable. This means that your custom authenticator must have no non-serializable members and also declare a serialVersionUID version number, as recommended in the Serializable Interface Java documentation.

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 WorkLightAuthLoginModule interface extends the java.io.Serializable interface, so as a result, your custom login module implements java.io.Serializable. This means that your login module must have no non-serializable fields and also declare a serialVersionUID version number, as recommended in the Serializable Interface Java documentation.

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, the handleChallenge 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 MobileFirst framework that the authentication finished successfully. The MobileFirst framework then automatically issues the original request that triggered the authentication.
   - submitFailure: This method notifies the MobileFirst framework that the authentication process failed. The MobileFirst 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 MobileFirst 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. Add the generated `serialVersionUID` constant to the class.
  3. 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, collect the credentials, 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, HttpServletResponse response,
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.length() > 0){
            this.username = request.getParameter("username");
        this.password = request.getParameter("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\":\"Please enter username and password\"}");
            return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
        }
    }

    if (!isAccessToProtectedResource)
        return AuthenticationResult.createFrom(AuthenticationStatus.REQUEST_NOT_RECOGNIZED);
  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_REQUIRED);
}
```

- 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, ServletException {
  response.setContentType("application/json; charset=UTF-8");
  response.setHeader("Cache-Control", "no-cache, must-revalidate");
  response.getWriter().print("{\"authStatus\":\"required\", \"errorMessage\":\"" + errorMessage + "\"}");
  return AuthenticationResult.createFrom(AuthenticationStatus.CLIENT_INTERACTION_REQUIRED);
}
```

- 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<String> roles, Map<String,
Object> attributes, Object credentials)
```

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, customAttributes, PASSWORD);
  return identity;
}
```

Coding the client side

Follow these steps:

1. Create a MobileFirst 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 the Custom authentication in hybrid applications tutorial.

IBM MobileFirst Platform Foundation for iOS provides other custom authentication mechanisms:

- Adapter-based custom authentication. For more information, see Implementing adapter-based authenticators.
- HTTP-based custom authentication. For more information, see HTTP-based authentication.

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 enables you to develop custom authentication logic by using a JavaScript function within a MobileFirst adapter.

Adapter-based authentication is flexible and customizable. The following diagram illustrates one possible implementation. The process is illustrated and described as follows.

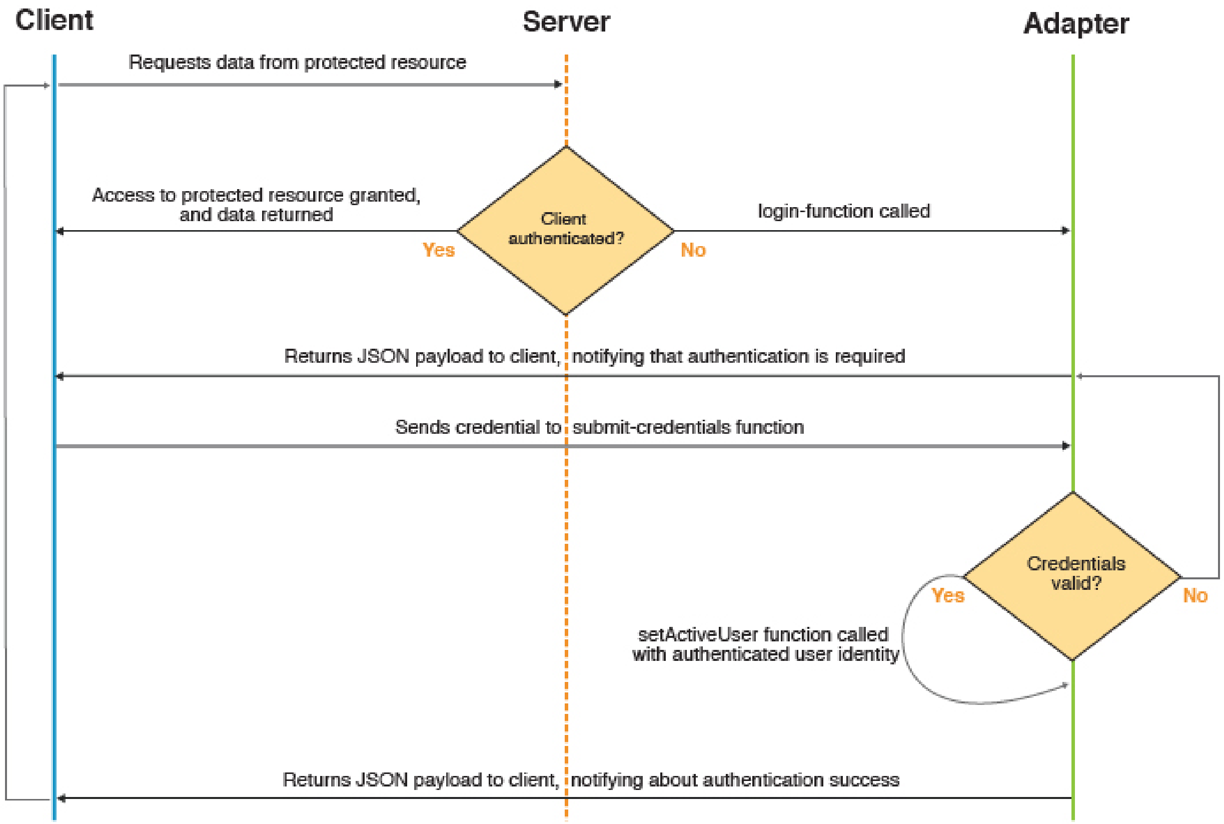

Client
Server
Adapter
Requests data from protected resource
Access to protected resource granted, and data returned
Yes
Client authenticated?
No
login-function called
Returns JSON payload to client, notifying that authentication is required
Sends credential to submit-credentials function
Credentials valid?
Yes
No
setActiveUser function called with authenticated user identity
Returns JSON payload to client, notifying about authentication success

Figure 8-39. Adapter-based authentication process

1. The client makes a request to the resource that is protected by adapter authentication.
2. MobileFirst 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 called.
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 8-253.

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 MobileFirst 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 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 IBM MobileFirst 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 fully qualified name of the Java class for adapter authenticators is `com.worklight.integration.auth.AdapterAuthenticator`. It takes the mandatory **`login-function`** parameter and the optional **`logout-function`** parameters. 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` in your `adapter.js` source file. In the example, these parameters are implemented as `AuthAdapter.onAuthRequired` and `AuthAdapter.onLogout`.

Note:

- Both `login-function` and `logout-function` should only be used internally by a MobileFirst 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 that 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 a call to `WL.Client.connect` before the adapter is called.
- If your MobileFirst Server runs on WebSphere Application Server, version 7 (any release) or releases of WebSphere Application Server Liberty 8.5 Before 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 for WebSphere Application Server or APAR PM79934 for WebSphere Application Server Liberty.

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 MobileFirst 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 MobileFirst 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`. For more information, see APAR PM79934.

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 MobileFirst 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.Logger.debug("Logged out");
}
```

Code the client side authentication

Perform the following steps:

1. Create a MobileFirst 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 MobileFirst framework that the authentication process completed successfully. The MobileFirst framework then automatically issues the original request that triggered authentication.
- The submitFailure function notifies the MobileFirst framework that the authentication process completed with failure. The MobileFirst 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 adapter-based authentication, see the tutorials on the Getting Started page.

IBM MobileFirst Platform Foundation for iOS provides other custom authentication mechanisms:

- Java-based custom authentication. For more information, see Implementing Java-based authenticators.
- HTTP-based custom authentication. For more information, see HTTP-based custom authentication.

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 MobileFirst project. To use it, remove the comments first.

This authenticator can be used with the `WASLTPAModule` login module.

Class Name

`com.worklight.core.auth.ext.WebSphereFormBasedAuthenticator`

Parameters

The adapter authenticator class has the following parameters:

Parameter	Description
login-page	Mandatory. The login page URL relative to the web application context.
error-page	Optional. The error page URL relative to the web application context. If this parameter is not set, the URL from the `login-page` is also used for the `error-page`.
cookie-domain	Optional. A String such as `example.com`, which specifies the domain in which the LTPA SSO cookie applies. If this parameter is not set, no domain attribute is set on the cookie. The single sign-on is then restricted to the application server host name and does not work with other hosts in the same domain. **Note:** This parameter is deprecated. Define this parameter in the `<loginModule>` entry for the `WebSphereLoginModule` instead. If the parameter is defined in both places, the value in the `<loginModule>` entry takes precedence.
httponly-cookie	Optional. A String with a value of either `true` or `false`, which specifies whether the cookie has the HttpOnly attribute set. This attribute helps to prevent cross-site scripting attacks. **Note:** This parameter is deprecated. Define this parameter in the `<loginModule>` entry for the `WebSphereLoginModule` instead. If the parameter is defined in both places, the value in the `<loginModule>` entry takes precedence.
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>
```

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.

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)` .

Device SSO in the OAuth-based security model

OAuth resources are not protected by security tests and do not have a single defined user realm per resource, therefore the standard MobileFirst device SSO behavior does not apply for them.

However, in order to obtain an access token in the MobileFirst OAuth-based security model, the client is also required to pass the application's security test. If SSO is configured for this security test, it will function for the MobileFirst OAuth-based security model as it does for the classical MobileFirst security model.

Configuring device single sign-on

Single sign-on (SSO) is a property of a login module. You can enable single sign-on for custom security tests and for mobile security tests.

About this task

You can enable single sign-on from a `<mobileSecurityTest>` element or from a `<loginModule>` element of the`authenticationConfig.xml` configuration file. For custom security tests, you enable single sign-on on the `<loginModule>` element. For mobile security tests, you enable single sign-on on the `testUser` realm of the `<mobileSecurityTest>` element.

Basically, you configure SSO in the same way for native IOS applications as for hybrid applications. However, for native SSO to work on iOS, this additional step is mandatory: In Xcode, add a Keychain Access Group with the same name for all apps that participate in device SSO.

Procedure

Take the following points into consideration, depending on how you choose to configure device single sign-on:

- When you configure <mobileSecurityTest> elements, enable single sign-on from the <securityTest> element by setting the value of the **sso** attribute to true. 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 you configure <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 that is 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 you configure <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 you configure 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 MobileFirst configuration properties, see “Configuration of MobileFirst applications on the server” on page 11-50.

Results

Device single sign-on implementations are successful if they conform to any of the following valid configurations. Avoid inconsistent states that result from configurations with built-in conflicts. Inconsistent states can result in the MobileFirst 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="WLDeviceAutoProvisioningLoginModule">
        <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. Such conflict 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="WLDeviceAutoProvisioningLoginModule">
        <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. In this example, the same realm contains 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="SampleAppRealm" sso="true"/>
    </mobileSecurityTest>
</securityTests>
```

Here is 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

When you use a reverse proxy, more configuration and settings are necessary. 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 device SSO to delegate user authentication to the MobileFirst Device SSO realm. For more information about the required configurations and samples, see IBM Security Access Manager for IBM MobileFirst Platform Foundation for iOS.

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 8-292.

About this task

Device SSO and reverse proxies

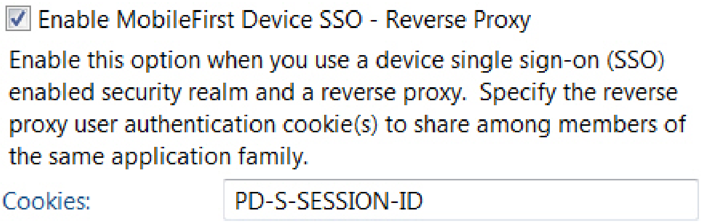
Device single sign-on with a reverse proxy can also be achieved with the “Simple data sharing” on page 8-246 feature. The Simple Data Sharing feature allows a set of applications to share authentication cookies that allow access through the reverse proxy and delegate authentication to the MobileFirst Server Device SSO realm.

With the Simple Data Sharing feature, you can tell the MobileFirst client runtime environment to share credentials among applications in the same MobileFirst 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 8-249.

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 8-247.
2. For hybrid applications, follow these steps.
   a. Ensure that you select the MobileFirst device SSO option.
   b. Specify a comma-separated list of cookie names that you want IBM MobileFirst Platform Foundation for iOS to remember and share among the applications in your specified family.



☑ Enable MobileFirst Device SSO - Reverse Proxy

Enable this option when you use a device single sign-on (SSO) enabled security realm and a reverse proxy. Specify the reverse proxy user authentication cookie(s) to share among members of the same application family.

Cookies: PD-S-SESSION-ID

3. For native applications, follow these steps.
   a. Add the `wlShareCookies` property in the MobileFirst properties file.
   b. Specify a comma-separated list of cookie names that you want IBM MobileFirst Platform Foundation for iOS to remember and share among the applications in your specified family.

```
wlShareCookies = PD-S-SESSION-ID
```

   Each application in the MobileFirst 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.

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 V6.2.0, developers that use MobileFirst client-side APIs who want to capture both platform (IBM MobileFirst Platform Foundation for iOS) 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 MobileFirst client libraries include a logger in JavaScript 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 MobileFirst 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 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.
- In production, the default log level is FATAL.
  - On iOS, production mode means that the DEBUG macros is not defined.
- 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.
- Log configuration set at the server by the MobileFirst administrator is piggybacked on responses to explicit `WLClient connect` and `invokeProcedure` API calls, and is applied automatically.

- All captured log data, if any, is sent to the MobileFirst Server during each successful client network `init` sequence and `invokeProcedure` response, with a 60 second minimum interval between sends.
  - 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 MobileFirst Server.

Note: For example, when capture is `ON` and the logger level is configured to `FATAL`, the logger captures uncaught exceptions and only logs at the FATAL level. It would not capture logs at ERROR, WARN, or INFO that may lead up to that failure. Alternatively, a more verbose logger level would also capture non-FATAL level logs leading up to the FATAL entry.

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 IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Operations 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 MobileFirst 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 and only logs at the FATAL level. It will not capture logs at ERROR, WARN, or INFO that may lead up to that failure. Alternatively, a more verbose logger level will also capture non-FATAL level logs leading up to the FATAL entry.

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 MobileFirst 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 MobileFirst administrator sets in the **Catalog** tab of the MobileFirst Operations 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:

- `applicationDidBecomeActive` (iOS)

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 MobileFirst code?

If your application code is making good use of the MobileFirst 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:

-
  ```
  // iOS
  OCLoggerDebugWithPackage(@"MyPackage", @"this is a debug message");
  // or Info, Log, Warn, and so on
  ```

  or

  ```
  // iOS
  OCLogger* logger = [OCLogger getInstanceWithPackage:@"MyPkg"];
  [logger debug:@"this is a debug message"];
  // or Info, Log, Warn, and so on
  ```

Then, set the filters on the logger to allow logging only for your package or packages:

-
  ```
  // 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 **Client Log Profiles** tab of the MobileFirst Operations Console.

Configuring the MobileFirst Logger

You can configure how the MobileFirst Logger runs on a range of client operating systems by modifying `WL.Logger` methods.

For more information on Logger and how to configure it, see Client-side log capture.

Set log level after IBM MobileFirst Platform Foundation for iOS starts

Logger is always on, but you can set the log level after IBM MobileFirst Platform Foundation for iOS starts.

```
WL.Client.init({
  onSuccess : function() {
    WL.Logger.config({'level' : 'fatal'});
  },
  onFailure : function(err){
    WL.Logger.error('Caught an exception', err);
  }
});
```

Select log levels

You can select from among various log levels.

Debug

Add a debug message.

```
WL.Logger.debug('Loop finished');
  // Loop finished.
```

Log

Add a log message.

```
WL.Logger.log('Got', response.statusCode, 'from server.');
  // Got 200 from the server.
```

Info

Add an informative message.

```
WL.Logger.info('Public IP address is', getIpAddress());
  // Public IP address is 192.168.1.102.
```

Warn

Add a warning message.

```
if (!window.indexedDB) {
  WL.Logger.warn('IndexedDB not supported, falling back to LocalStorage.');
    // IndexedDB not supported, falling back to LocalStorage.
}
```

Error

Add an error message.

```
try {
  // Code that may throw new Error('Something failed here.').
} catch (e) {
  WL.Logger.error('Caught an exception', e);
    // Caught an exception Error: Something failed here.
  }
```

Log different data types

You can log a range of data types including numbers, strings, and arrays

Strings

```
WL.Logger.info('Hello', 'world.');
  //Hello World.
```

Booleans

```
WL.Logger.info(true, false);
  //true false
```

Numbers

```
 WL.Logger.info(1,2,3.14,4,5,6,7);
  //1 2 3.14 4 5 6 7
```

Arrays

```
WL.Logger.info([1,2,3], [[1,2,3], [1,2,3]])
  //[1,2,3] [[1,2,3], [1,2,3]]
```

Objects

```
WL.Logger.info({hello: 'world'}, {hey: {test: [1,2,3, {hello: 'world'}]}});
  //{"hello" : "world"} {"hey" : {"test": [1,2,3, {"hello": "world"}]}}
```

Exceptions

```
var e = new Error('Something failed');
  var te = new TypeError('Wrong type');
  WL.Logger.info(e, te);
  //Error: Something failed TypeError: Wrong type
```

undefined

```
 var undef;
  WL.Logger.info(undefined, undef);
  //undefined undefined
```

null

```
 var n = null;
  WL.Logger.info(null, n);
  //null null
```

Any Combination

```
WL.Logger.info('Hey', 1, 2, true, false, [1,2,3], {hey: 'world'}, new Error('Uh oh'), undefined, null);
 //Hey 1 2 true false [1,2,3] {"hey": "world"} Error: Uh oh undefined null
```

Filter log levels

You can filter logs to display only logs of equal or lower priority, set by the level property.

For example, to show only warn, error, and fatal logs:

```
WL.Logger.config({level: ['warn']});
```

To show only error and fatal logs:

```
WL.Logger.config({level: ['error']});
```

To show all logs:

```
WL.Logger.config({level : []});
```

To set Logger priority, you can specify a string or int value:

```
WL.Logger.config({level: 'trace'});
WL.Logger.config({level: 600});
```

Possible string values:

```
'trace',
'debug',
'log',
'info',
'warn',
'error',
'fatal'
'analytics'
```

Note: These values are not case-sensitive, for example LOG and Log are also possible values.

Possible int values:

```
25  (analytics)
50  (fatal)
100 (error)
200 (warn)
300 (info)
400 (log)
500 (debug)
600 (trace)
```

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.config({filters : {'wl.jsonstore' : 'info'});
```

Associate a log message with a package, and log a message. To exclude packages from logging (blacklist), exclude them from the filter object:

```
var JSONSTORE_PKG = 'wl.jsonstore';

WL.Logger.info('Hey!'); // Ignored.

WL.Logger.ctx({pkg: JSONSTORE_PKG}).info('JSONStore started');
// JSONStore started.

WL.Logger.ctx({pkg: JSONSTORE_PKG}).warn('JSONStore finished executing find.');
// JSONStore finished executing find.

WL.Logger.ctx({pkg: 'wl.analytics'}).warn('Hello.'); // 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: {level: false, pkg: true}, android: false }
```

The list of keys returned match the options that you can pass to **WL.Logger.config**.

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.
```

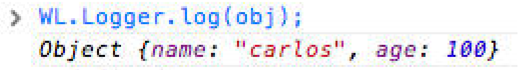
Stringify

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.

By default, all logs are stringified. To disable this feature:

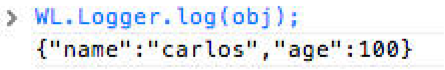
```
WL.Logger.config({stringify: false});
```



```
> WL.Logger.log(obj);
  Object {name: "carlos", age: 100}
```

To re-enable this feature:

```
WL.Logger.config({stringify: true});
```



```
> WL.Logger.log(obj);
  {"name":"carlos","age":100}
```

Callback

You can pass a callback function to WL.Logger.config that is called after every log message.

The callback function takes the following 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 by 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.config({callback: ajaxSender});
```

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.config({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.config({tag: {level: false, pkg: false} });

WL.Logger.info('Hello');
// Hello

WL.Logger.ctx({pkg: 'MYPKG'}).error('Hey');
// Hey
```

Note: Tags only show in web applications and not in hybrid applications.

Method chaining

You can invoke multiple method calls by chaining logger methods.

You can chain these logger methods:

```
WL.Logger.ctx
WL.Logger.create
```

This example carries out these steps:

- Sets the package context to `com.my.app`
- Logs `Hello`.

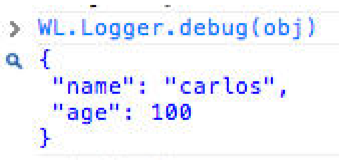
```
WL.Logger.ctx({pkg: 'com.my.app'}).log('Hello');
// '[com.my.app] Hello'
```

Pretty-print JSON objects

You can format JSON objects by enabling `stringify`.

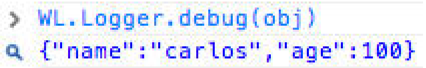
By enabling (pretty: true) you can display JSON objects in a more readable format.

```
var obj = {name: 'carlos', age: 100};
WL.Logger.config({stringify: true, pretty: true});
```



```
> WL.Logger.debug(obj)
  {
   "name": "carlos",
   "age": 100
  }
```

```
WL.Logger.config({pretty: false});
```



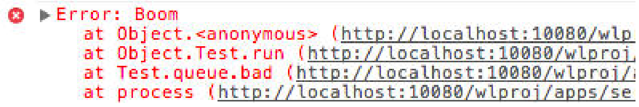
```
> WL.Logger.debug(obj)
  {"name":"carlos","age":100}
```

Print stack traces

You can print stack traces for certain objects.

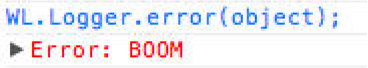
You can print stack traces for an object if the object is an instance of Error (if (object instance of Error) evaluates true).

```
WL.Logger.config({stringify: true, stacktrace : true});
WL.Logger.error("BOOM");
```



```
▶Error: Boom
    at Object.<anonymous> (http://localhost:10080/wlp
    at Object.Test.run (http://localhost:10080/wlproj
    at Test.queue.bad (http://localhost:10080/wlproj/
    at process (http://localhost:10080/wlproj/apps/se
```

```
WL.Logger.config({stacktrace: false});
WL.Logger.error(object);
```



```
WL.Logger.error(object);
▶Error: BOOM
```

Logger Android check and override

Logger checks the operating system on which it is running to determine whether to use the Android logger. You can override this behavior.

By default, WL.Logger checks the operating system at run time, and if it is running on Android it attempts to use the cordova plug-in. If the plug-in fails, it falls back to console.log. There are several differences between the cordova plug-in logger and console log:

Cordova plugin logger

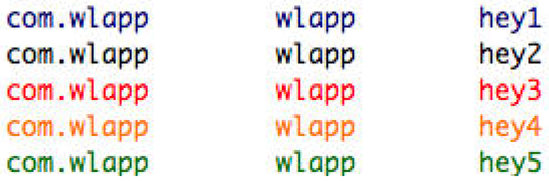
Asynchronous

Provides better output in LogCat

Requires that the deviceready event previously fired.

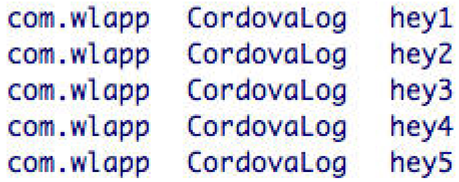
Console log
Synchronous

Native logger + LogCat:



```
com.wlapp        wlapp        hey1
com.wlapp        wlapp        hey2
com.wlapp        wlapp        hey3
com.wlapp        wlapp        hey4
com.wlapp        wlapp        hey5
```

console.log + LogCat:



```
com.wlapp   CordovaLog   hey1
com.wlapp   CordovaLog   hey2
com.wlapp   CordovaLog   hey3
com.wlapp   CordovaLog   hey4
com.wlapp   CordovaLog   hey5
```

To override the Android check, do one of these:

- Pass `android: false` to `WL.Logger.config`

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);
```

As examples, you can use environment-specific settings to specify no logs in production and default logging for non-production:

```
//Change accordingly
var CURRENT_ENV = 'production';

//General init options
var wlInitOptions = {};

//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

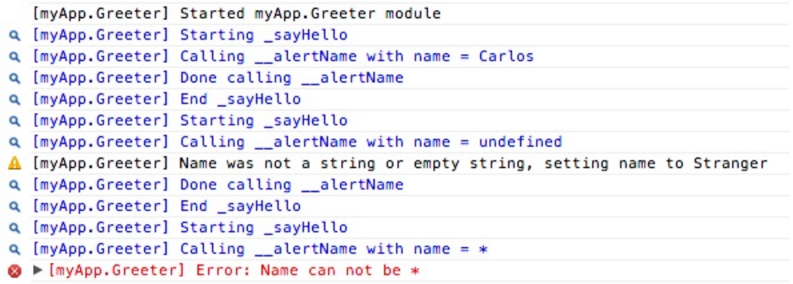
```
var myApp = myApp || {};
myApp.Greeter = (function (WL) {

    //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 8-40. Log output

Server preparation for uploaded log data

You must prepare your server to receive uploaded client log data.

Upon receiving uploaded client logs, the MobileFirst 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 MobileFirst Server; you must install and configure them. To receive and persist uploaded client logs at the MobileFirst Server, you must take at least one of the following two actions:

1. Install the IBM MobileFirst Platform Operational Analytics as described in "Installing the IBM MobileFirst Platform Operational Analytics" on page 6-191.
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 8-94.

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 MobileFirst 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, there is no security that protects the `loguploader` servlet that receives uploaded client logs and analytics at the MobileFirst Server. You can configure the security tests that protect the servlet in the authenticationConfig.xml file. 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, the 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 MobileFirst production servers.

Client-side log capture configuration from MobileFirst Operations Console

In the redesigned MobileFirst Operations Console, administrators can use the Client Log Profile link of a runtime environment to adjust 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 MobileFirst administrator creates a configuration profile, the log configuration is concatenated with responses to explicit WLClient connect and invokeProcedure API calls, and is applied automatically.

When the MobileFirst administrator removes a configuration profile, on the next client application WLClient connect and invokeProcedure API calls, the client reverts to its configuration before the server configuration profile override.

What is provided on the client side

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 10-2.

Migrating applications created on IBM Bluemix to IBM MobileFirst Platform Foundation for iOS

You can migrate native applications that were developed on Bluemix Mobile Services to IBM MobileFirst Platform Foundation for iOS.

The existing Bluemix Mobile Services app must use a custom identity provider. To provide this security for the app when it is running on IBM MobileFirst Platform Foundation for iOS, you must modify some server-side components.

To migrate the application to IBM MobileFirst Platform Foundation for iOS, you must complete the following tasks:

- Adapt the server-side authentication code, by using one of the following HTTP-based custom authentication methods:
  - Security realm within a Java adapter
  - Oauth TAI
  - A protected Node.js resource
- Migrate the client application. Migrating the client application includes the following steps:
  - Installing the IBM MobileFirst Platform Foundation for iOS SDK, including its associated framework. For iOS applications, you also install a compatibility framework.
  - Making configuration changes to your Xcode project
  - Modifying the application code to remove references that are specific to the Bluemix Mobile Services environment, replacing them with references required in the IBM MobileFirst Platform Foundation for iOS environment

For iOS applications, an API, called the compatibility API, translates Bluemix Mobile Services calls to IBM MobileFirst Platform Foundation for iOS calls. The compatibility framework communicates with the MobileFirst iOS client framework, which in turn communicates with the MobileFirst Server.

Migration scenario

To illustrate how to migrate an existing Bluemix Mobile Services app to IBM MobileFirst Platform Foundation for iOS, this documentation uses an existing Bluemix application that is written in Java. It runs on the Liberty for Java runtime and uses Oauth TAI for security. The Oauth TAI security is implemented as a custom identity provider through the Bluemix Mobile Client Access service.

The following code is the JAX-RS Java class that is deployed to the Liberty for Java server runtime on Bluemix. The class has one service (`/hello`) and a custom identity provider that is defined by the following HTTP APIs:

- `/startAuthorization`
- `/handleChallengeAnswer`

```
package com.mycompany;

import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import com.ibm.json.java.JSON;
import com.ibm.json.java.JSONObject;

@Path("/")
public class CustomAuthResource {

  // local user repository
  private final static String USER_STORE_JSON = "{\"leonard\": {\"password\": \"1234\", \"displayNam

  // failure JSON
  private final static String FAILURE_JSON = "{\"status\": \"failure\"}";

  // challenge JSON
  private final static String CHALLENGE_JSON = "{\"status\": \"challenge\", \"challenge\": {\"messag

  @POST
  @Consumes("application/json")
  @Path("/{tenantId}/customAuthRealm_3/startAuthorization")
  @Produces(MediaType.APPLICATION_JSON)
  public JSONObject startAuthorization(String payload,
 @PathParam("tenantId") String deviceId,
 @PathParam("realmName") String realmName) throws Exception {

 JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
 return returnJson;
  }

  @POST
  @Consumes("application/json")
  @Path("/{tenantId}/customAuthRealm_3/handleChallengeAnswer")
  @Produces(MediaType.APPLICATION_JSON)
  public JSONObject handleChllengeAnswer(String payload,
 @PathParam("tenantId") String deviceId,
 @PathParam("realmName") String realmName) throws Exception {

    JSONObject userStoreJson = (JSONObject) JSON.parse(USER_STORE_JSON);
 JSONObject failedResponseJson = (JSONObject) JSON.parse(FAILURE_JSON);

    if (payload == null || payload.isEmpty()) {
      return failedResponseJson;
    }

    JSONObject payloadJson = (JSONObject) JSON.parse(payload);
    JSONObject challengeAnswer = (JSONObject) payloadJson.get("challengeAnswer");

    if (challengeAnswer == null) {
      return failedResponseJson;
    }

    String userName = (String) challengeAnswer.get("userName");
    String password = (String) challengeAnswer.get("password");

    if (userName == null || userName.isEmpty() || password == null
      || password.isEmpty()) {

      return failedResponseJson;
    }

    if (userStoreJson.containsKey(userName)) {
      JSONObject userInfoJson = (JSONObject) userStoreJson.get(userName);
      String userPassword = (String) userInfoJson.get("password");
      String userDisplayName = (String) userInfoJson.get("displayName");

      if (password.equals(userPassword)) {
        JSONObject returnJson = new JSONObject();
        JSONObject userIdentityJson = new JSONObject();
        userIdentityJson.put("userName", userName);
        userIdentityJson.put("displayName", userDisplayName);
        returnJson.put("status", "success");
        returnJson.put("userIdentity", userIdentityJson);

        return returnJson;

      }
    }
 return failedResponseJson;
  }

  @GET
  @Path("/{tenantId}/hello")
  public String hello() {
    return "Hello from JAX-RS";
  }
}
```

Adapting the server-side code

For an existing Bluemix Mobile Services application to run on premises, on IBM MobileFirst Platform Foundation for iOS, you must modify parts of the MobileFirst Server runtime.

Migrating a Bluemix app that uses HTTP-based custom authentication to on-premises MobileFirst Server

If your application runs on Liberty for Java on Bluemix, you can implement a Java adapter on the on-premises MobileFirst Server to provide HTTP-based custom authentication for your migrated application.

Before you begin

- You must have the MobileFirst Server installed.
- You must have the IBM MobileFirst Platform Command Line Interface for iOS installed.
- You must have created a native application for iOS project on your on-premises MobileFirst Server. For more information, see “Developing native applications” on page 8-7.

Procedure

1. Configure a realm name and a login module in the `authenticationConfig.xml` file. The `authenticationConfig.xml` file is in the `/server/conf` directory of your new MobileFirst project. Use the same realm name that was used on Bluemix . For example:

```
<!-- Under realms node -->
<realm name="customAuthRealm_3" loginModule="customAuthLoginModule_3">
  <className>com.worklight.core.auth.ext.CustomIdentityAuthenticator</className>
  <parameter name="providerUrl" value="http://localhost:10080/CustomAuthProject/adapters/custo
</realm>

<!-- Under loginModules node -->
<loginModule name="customAuthLoginModule_3">
  <className>com.worklight.core.auth.ext.CustomIdentityLoginModule</className>
</loginModule>
```

2. Optional: If you want your application to authenticate with the accessing mobile device by using the user identity, edit the `application-descriptor.xml` file and set the user identity realm to one of the realm names that is defined in the `authenticationConfig.xml` file.
3. Implement the Java adapter. For more information about Java adapters, see “MobileFirst Java adapters” on page 8-24. For client-side examples of implementing the challenge handler for an app with a Java adapter, see Adapter-based authentication.
   a. Copy the code of the JAX-RS class that is deployed to the Liberty for Java runtime on IBM Bluemix and paste it into the file that implements the Java adapter.
   b. Modify the code of the newly created Java adapter's JAX-RS Resource class:
      1) Remove the variable `/{tenantId}/` from the resource PATH.
      2) Remove strings that contain `@PathParam`. For example, you would remove the strings `@PathParam("tenantId") String deviceId` and `@PathParam("realmName") String realmName` from the following code:

```
@POST

@Consumes ("application/json")
@Path("/{tenantId}/customAuthRealm_3/startAuthorization")
@Produces(MediaType.APPLICATION_JSON)
    public JSONObject startAuthorization(String payload,
            @PathParam("tenantId") String deviceId,
            @PathParam("realmName") String realmName) throws Exception {
        JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
        return returnJson;
    }
```

      3) Add a scope. In the following example, the scope `@OAuthSecurity (scope="customAuthRealm_3")`, is added just before the hello() method.

   The following example shows a JAX-RS Resource class after modification:

```
@Path("/")
public class CustomAuthResource {
  //local user repository
  private final static String USER_STORE_JSON = "{\"leonard\": {\"password\": \"1234\", \"dis

  //failure JSON
  private final static String FAILURE_JSON = "{\"status\": \"failure\"}";

  //challenge JSON
  private final static String CHALLENGE_JSON = "{\"status\": \"challenge\", \"challenge\": {\

  @POST
  @Consumes ("application/json")
  @Path("/customAuthRealm_3/startAuthorization")
  @Produces(MediaType.APPLICATION_JSON)

  public JSONObject startAuthorization(String payload) throws Exception {
    JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
    return returnJson;
  }

  @POST
  @Consumes ("application/json")
  @Path("/customAuthRealm_3/handleChallengeAnswer")
  @Produces(MediaType.APPLICATION_JSON)
  public JSONObject handleChallengeAnswer(String payload) throws Exception {
    JSONObject userStoreJson = (JSONObject) JSON.parse(USER_STORE_JSON);
    JSONObject failedResponseJson = (JSONObject) JSON.parse(FAILURE_JSON);

    if(payload == null || payload.isEmpty()) {
      return failedResponseJson;
    }

    JSONObject payloadJson = (JSONObject) JSON.parse(payload);
    JSONObject challengeAnswer = (JSONObject) payloadJson.get("challengeAnswer");
    if (challengeAnswer == null ) {
      return failedResponseJson;
    }

    String userName = (String) challengeAnswer.get("userName");
    String password = (String) challengeAnswer.get("password");

    if (userName == null || userName.isEmpty() || password == null || password.isEmpty()) {
      return failedResponseJson;
    }

    if (userStoreJson.containsKey(userName)) {
      JSONObject userInfoJson = (JSONObject) userStoreJson.get(userName);
      String userPassword = (String) userInfoJson.get("password");
      String userDisplayName = (String) userInfoJson.get("displayName");

      if (password.equals(userPassword)) {
        JSONObject returnJson = new JSONObject();
        JSONObject userIdentityJson = new JSONObject();
        userIdentityJson.put("userName", userName);
        userIdentityJson.put("displayName", userDisplayName);
        returnJson.put("status", "success");
        returnJson.put("userIdentity", userIdentityJson);

        return returnJson;
      }
    }
    return failedResponseJson;
  }

  @GET
  @OAuthSecurity (scope="customAuthRealm_3")
  @Path("/hello")
  public String hello(){
    return "Hello from JAX-RS";
  }
}
```

What to do next

1. Deploy the adapter to the MobileFirst Server. For more information, see “Deploying adapters” on page 8-106.
2. Migrate the client application. For more information, see “Adapting the iOS client application” on page 9-11.

Migrating a Bluemix app that uses an OAuth TAI filter to on-premises MobileFirst Server

If your application runs on Liberty for Java on Bluemix, you can run it on the on-premises MobileFirst Server and protect resources by adding the built-in OAuth TAI (Trust Association Interceptor) filter.

Before you begin

- You must have the MobileFirst Server installed. The underlying application server must be WebSphere Application Server Liberty. For step-by-step instructions for installing WebSphere Application Server Liberty, see “Tutorial for a basic installation of MobileFirst Server” on page 6-24.

Important:

– To use the OAuth TAI authentication method, your underlying application server can also be WebSphere Application Server. However, the following instructions assume that the application server is WebSphere Application Server Liberty unless otherwise stated. For instructions on installing WebSphere Application Server, see the installation instructions for your version of WebSphere Application Server in the IBM Knowledge Center.

– In the `server.xml` file in your WebSphere Application Server Liberty, make sure you have added a `<feature>appSecurity-2.0</feature>` element, and make sure the `<basicRegistry>` element exits.

- You must have the IBM MobileFirst Platform Command Line Interface for iOS installed.
- You must have created a native application for iOS project on your on-premises MobileFirst Server. For more information, see "Developing native applications" on page 8-7.

About this task

For more information about OAuth TAI configuration, see "Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty" on page 8-199.

Procedure

1. Configure a realm name and a login module in the `authenticationConfig.xml` file. The `authenticationConfig.xml` file is in the `/server/conf` directory of your new MobileFirst project. Use the same realm name that was used on Bluemix . For example:

```
<!- Under realms node -->
<realm name="customAuthRealm_3" loginModule="customAuthLoginModule_3">

  <className>com.worklight.core.auth.ext.CustomIdentityAuthenticator</className>

  <parameter name="providerUrl" value="http://localhost:9080/CustomAuthProject/adapters/customAut
</realm>

<!- Under loginModules node -->
<loginModule name="customAuthLoginModule_3">
  <className>com.worklight.core.auth.ext.CustomIdentityLoginModule</className>
</loginModule>
```

2. Change the path for the following endpoint protocols that must be exposed: `/startAuthorization` and `/handleChallengeAnswer`. Change the path from what it was on Bluemix:

   /apps/${*tenant id*}/${*realm_name*}/${*endpoint_api*}

   to what is required for IBM MobileFirst Platform Foundation for iOS:

   /${*realm_name*}/${*endpoint_api*}

   For example:

   `/apps/{tenantId}/customAuthRealm_3/startAuthorization`

   changes to

   `/customAuthRealm_3/startAuthorization`

   and

   `/apps/{tenantId}/customAuthRealm_3/handleChallengeAnswer` changes to

```
/customAuthRealm_3/handleChallengeAnswer
```

3. Optional: If you want your application to authenticate with the accessing mobile device by using the user identity, edit the `application-descriptor.xml` file and set the user identity realm to one of the realm names that is defined in the `authenticationConfig.xml` file.
4. Modify the JAX-RS Resource class code for the application:
   a. Remove the variable `/{tenantId}/` from the resource PATH.
   b. Remove strings that contain `@PathParam`. For example, you would remove the strings `@PathParam("tenantId") String deviceId` and `@PathParam("realmName") String realmName` from the following code:

   ```
   @POST

   @Consumes ("application/json")
   @Path("/{tenantId}/customAuthRealm_3/startAuthorization")
   @Produces(MediaType.APPLICATION_JSON)
       public JSONObject startAuthorization(String payload,
               @PathParam("tenantId") String deviceId,
               @PathParam("realmName") String realmName) throws Exception {
           JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
           return returnJson;
       }
   ```
5. Export the updated application as a `WAR` file.
6. Place the `WAR` file in the *`usr`*`/servers/`*`serverName`*`/apps` folder to deploy it to WebSphere Application Server Liberty.
7. Configure WebSphere Application Server Liberty for OAuth TAI. For more information about configuring WebSphere Application Server Liberty for OAuth TAI, see Installing OAuthTAI in WebSphere Application Server Liberty in “Protecting resources on WebSphere Application Server or WebSphere Application Server Liberty” on page 8-199.
   a. Edit the `server.xml` file to add the OAuth TAI feature. Note that the security role `TAIUserRole` is mapped to a special subject named `ALL_AUTHENTICATED_USERS`. For example:

   ```
   <usr_OAuthTAI id="myOAuthTAI">
      <securityConstraint httpMethods="GET POST" securedURLs="/worklight-oauth-java-tests/*"
          scope="wl:Realm1 wl:Realm2"/>
      <securityConstraint httpMethods="DELETE" securedURLs="/another-context-root/*"/>
   </usr_OAuthTAI>

       <basicRegistry id="basic" realm="BasicRealm" />

       <application type="war" id="basicauth" name="basicauth" location="${server.config.dir}/
       <application-bnd>
       <security-role name="TAIUserRole">
        <special-subject type="ALL_AUTHENTICATED_USERS" />
        </security-role>
       </application-bnd>
      </application>
   ```
   b. Modify the server environment variable. In the `server.env` file, change the existing URL:

   ```
   imfServiceUrl=http://imf-authserver.ng.bluemix.net/imf-authserver
   ```

   to

   `publicKeyServerUrl=`*`authorization_server_URL`*

   where

authorization_server_URL
is the URL to the MobileFirst authorization server.

8. Modify the `server.xml` file.
   a. Remove `realmName="imfRealm"`.
   b. Replace the separator "," with a space if it exists in the values of the **httpMethods** or the **securedURLs** attributes. The following code snippet shows an example of code before modification:

```
<usr_OAuthTAI id="myOAuthTAI" realmName="imfRealm">
   <securityConstraint httpMethods="GET, POST" securedURLs="/custom-oauth-java/*"/>
</usr_OAuthTAI>
```

   The following code snippet shows the code after modification is complete:

```
<usr_OAuthTAI id="myOAuthTAI">
  <securityConstraint httpMethods="GET POST" securedURLs="/custom-oauth-java/*"/>
</usr_OAuthTAI>
```

   c. Modify the **id**, **location**, and **name** attributes of the `application` element in the `server.xml` file based on the values for these attributes that are in your application.

Example

Here is an example of the revised code for the JAX-RS Resource class:

```
//local user repository
 private final static String USER_STORE_JSON = "{\"leonard\": {\"password\": \"1234\", \"displayNam
\"lisa\": {\"password\": \"1234\", \"displayName\": \"Lisa G.\"}}";

 //failure JSON
 private final static String FAILURE_JSON = "{\"status\": \"failure\"}";

 //challenge JSON
 private final static String CHALLENGE_JSON = "{\"status\": \"challenge\", \"challenge\": {\"messag
\"missing_credentials\"}, \"stateId\" : \"myStateId\"}";

@POST
@Consumes ("application/json")
@Path("/customAuthRealm_3/startAuthorization")
@Produces(MediaType.APPLICATION_JSON)
public JSONObject startAuthorization(String payload) throws Exception {
 JSONObject returnJson = (JSONObject) JSON.parse(CHALLENGE_JSON);
 return returnJson;
}

@POST
@Consumes ("application/json")
@Path("/customAuthRealm_3/handleChallengeAnswer")
@Produces(MediaType.APPLICATION_JSON)
public JSONObject handleChllengeAnswer(String payload) throws Exception {

 JSONObject userStoreJson = (JSONObject) JSON.parse(USER_STORE_JSON);
 JSONObject failedResponseJson = (JSONObject) JSON.parse(FAILURE_JSON);

 if(payload == null || payload.isEmpty()) {
  return failedResponseJson;
 }
 JSONObject payloadJson = (JSONObject) JSON.parse(payload);
 JSONObject challengeAnswer = (JSONObject) payloadJson.get("challengeAnswer");

 if (challengeAnswer == null ) {
  return failedResponseJson;
 }

 String userName = (String) challengeAnswer.get("userName");
```

```
    String password = (String) challengeAnswer.get("password");

    if (userName == null || userName.isEmpty() || password == null || password.isEmpty()) {
     return failedResponseJson;
    }

    if (userStoreJson.containsKey(userName)) {
     JSONObject userInfoJson = (JSONObject) userStoreJson.get(userName);
     String userPassword = (String) userInfoJson.get("password");
     String userDisplayName = (String) userInfoJson.get("displayName");

     if (password.equals(userPassword)) {
      JSONObject returnJson = new JSONObject();
      JSONObject userIdentityJson = new JSONObject();
      userIdentityJson.put("userName", userName);
      userIdentityJson.put("displayName", userDisplayName);

      returnJson.put("status", "success");
      returnJson.put("userIdentity", userIdentityJson);
      return returnJson;
     }
    }

    return failedResponseJson;
  }
```

What to do next

1. Restart the application server.
2. Migrate the client application. For more information, see “Adapting the iOS client application” on page 9-11.

Migrating a Bluemix app that uses a Node.js filter to on-premises MobileFirst Server

If your Bluemix app uses a Node.js filter, you can run it on the on-premises MobileFirst Server and protect Node.js resources by adding the built-in Node.js filter.

Before you begin

- You must have the MobileFirst Server installed.
- You must have the IBM MobileFirst Platform Command Line Interface for iOS installed.
- You must have created a native application for iOS project on your on-premises MobileFirst Server. For more information, see “Developing native applications” on page 8-7.

Procedure

1. Configure a realm name and a login module in the authenticationConfig.xml file. The authenticationConfig.xml file is in the /server/conf directory of your new MobileFirst project. Use the same realm name that was used on Bluemix . For example:

```
<!-- Under realms node -->
<realm name="customAuthRealm_3" loginModule="customAuthLoginModule_3">
  <className>com.worklight.core.auth.ext.CustomIdentityAuthenticator</className>
  <parameter name="providerUrl" value="http://localhost:3000/CustomAuthProject/adapters/custom
</realm>
```

```
<!-- Under loginModules node -->
<loginModule name="customAuthLoginModule_3">
  <className>com.worklight.core.auth.ext.CustomIdentityLoginModule</className>
</loginModule>
```

2. Optional: If you want your application to authenticate with the accessing mobile device by using the user identity, edit the `application-descriptor.xml` file and set the user identity realm to one of the realm names that is defined in the `authenticationConfig.xml` file.
3. Change the path for the following endpoint protocols that must be exposed: `/startAuthorization` and `/handleChallengeAnswer`. Change the path from what it was on Bluemix:

   ```
   /apps/${tenant id}/${realm_name}/${endpoint_api}
   ```

   to what is required for IBM MobileFirst Platform Foundation for iOS:

   ```
    /${realm_name}/${endpoint_api}
   ```

   For example:

   ```
   /apps/{tenantId}/customAuthRealm_3/startAuthorization
   ```

   changes to

   ```
   /customAuthRealm_3/startAuthorization
   ```

   and

   ```
   /apps/{tenantId}/customAuthRealm_3/handleChallengeAnswer
   ```

   changes to

   ```
   /customAuthRealm_3/handleChallengeAnswer
   ```

4. Replace the `require` declaration. Change this declaration from what it was on Bluemix:

   ```
   require('passport-imf-token-validation').$(any strategy)
   ```

   to what is required for IBM MobileFirst Platform Foundation for iOS:

   ```
    require('passport-mfp-token-validation').Strategy
   ```

5. Replace the protection part of each function with the following code:

   ```
   passport.authenticate('mobilefirst-strategy', {

                   session : false

               }),
   ```

Example

Here is an example of the code:

```
mfpStrategy = require('passport-mfp-token-validation').Strategy;

           ...

app.post('/customAuthRealm/startAuthorization',

           passport.authenticate('mobilefirst-strategy', {

               session : false

           }),     function(req, res) {

           var returnedJSON = startAuthorization(req.body.headers);
```

```
           res.json(returnedJSON);

       });


   app.post('/customAuthRealm/handleChallengeAnswer',

           passport.authenticate('mobilefirst-strategy', {

               session : false

           }),     function(req, res) {

           var returnJSON = handleChallengeAnswer(req.body.headers, req.body.stateId, req.body.ch

           res.json(returnJSON);

       });var startAuthorization = function(headers) {
       return {
           status: "challenge",
               challenge: {
                   message: "missing_credentials"
               },
           stateId : "my_custom_state_id"
       };
   };

var handleChallengeAnswer = function(headers, stateId, challengeAnswer) {
       console.log('State id ' + stateId);
       if (challengeAnswer && users[challengeAnswer.userName] && challengeAnswer.password === use
           return {
               status: "success",
               userIdentity: {
                   userName: challengeAnswer.userName,
                   displayName: users[challengeAnswer.userName].displayName
               }
           };
       } else {
           return {
               status: "failure"
           }
       }
   };
```

What to do next

1. Deploy the Node.js service.
2. Migrate the client application. For more information, see "Adapting the iOS client application."

Adapting the iOS client application

To adapt the client application, you must install new components, configure the Xcode project, and then modify the application code.

Installing required components

To prepare for migrating the client application, you must install the required MobileFirst SDK and iOS frameworks.

Before you begin

Your application must use CocoaPods for dependency management. Specifically,

- You must have CocoaPods, the dependency manager for Xcode projects, installed in your development environment. For more information, see the "Getting Started" guide for CocoaPods installation.
- Your application must be set up to use CocoaPods. For more information, see Using CocoaPods.
- You must have an existing `Podfile`. If one does not exist, create one by using the **`pod init`** command. For more information, see Using CocoaPods.

Procedure

1. Edit the `Podfile` file:
   a. Create a folder in your environment and copy your original Xcode project to it. Use this copied project for the migration.
   b. Open the `Podfile` file that is in the root directory of the copied project with a text editor.
   c. Comment out or remove references to dependencies that are associated with the IBM MobileFirst Platform for iOS SDK. For example:

      ```
      #pod 'IMFGoogleAuthentication'
      #pod 'IMFFacebookAuthentication'
      #pod 'IMFPush'
      #pod 'IMFData'
      #pod 'IMFURLProtocol'
      #pod 'IMFCore'
      ```

      For a list of all the possible dependencies, see the Installing the IBM MobileFirst Platform for iOS Client SDK using CocoaPods documentation on Bluemix.
   d. Add the following lines and save the changes:

      ```
      source 'https://github.com/CocoaPods/Specs.git'
      pod 'IMFCompatibility'
      ```
2. Open **Terminal** and navigate to the location of the `Podfile` file.
3. Verify that the Xcode project is closed.
4. Type `pod install` to run the **`pod install`** command. This command installs the following components and integrates them with the mobile application Xcode project:
   - the IBM MobileFirst Platform for iOS compatibility framework, called `IMFCompatibility.framework`
   - the MobileFirst SDK, called the `IBMMobileFirstPlatformFoundation.framework`

What to do next

Configure the Xcode project. See “Configuring the Xcode project.”

Configuring the Xcode project

You must change some settings and add a file to your Xcode project as part of the migration process.

Procedure

1. Open your *`ProjectName`*`.xcworkspace` file in Xcode by typing `open` *`ProjectName`*`.xcworkspace` from a command prompt. This file is located in the same directory as the *`ProjectName`*`.xcodeproj` file.
2. In Xcode, select **Build Settings**. In the **Other Linker Flags** field, enter `${inherited}`.

3. Add a `worklight.plist` file to the project. All IBM MobileFirst Platform Foundation for iOS applications require a `worklight.plist` file to be present in the application resources. A `.plist` template is supplied in the `IBMMobileFirstPlatformFoundation` pod. Add the `worklight.plist` file to your copied application by completing the following steps:
   a. In the `Pods` subdirectory of your project, click **Pods** > **IBMMobileFirstPlatformFoundation** > **Resources**. Locate the `sample.worklight.plist` file in the `Resources` directory.
   b. Copy the `sample.worklight.plist` file to the top-level directory of the project. For example, if the project name is `MyProject`, the original `sample.worklight.plist` file is in the directory `MyProject/Pods/IBMMobileFirstPlatformFoundation/Resources`. Create a copy of `sample.worklight.plist` and move the copy to the directory `MyProject`.
   c. Rename the copy of file `sample.worklight.plist` to `worklight.plist` and add it to the project.
4. Edit the `worklight.plist` file and set the **`application id`** key to the name of your application that is to be deployed to the IBM MobileFirst Platform Foundation for iOS server.

   Note: Make sure to use the same application name that you defined when you created theIBM MobileFirst Platform Foundation for iOS project.

What to do next

Modify the application code. See "Modifying the application code."

Modifying the application code

You must modify the code in your existing IBM MobileFirst Platform for iOS application on Bluemix as part of the migration process.

Before you begin

Complete the configuration steps that are described in "Installing required components" on page 9-11.

Procedure

1. In your application to be migrated, find and replace all of the references to `IMF/`*`string`* with references to `IMFCompatibility`.
   a. Search for all import directives that contain the string `#import <IMF`.
   b. In all instances of `#import <IMF/`*`string`*`/IMF`*`string`*`.h>`, replace *`string`* with `IMFCompatibility`,

      where

      `string` is any string appended to `#import <IMF` and to `/IMF`, such as `IMFCore` or `IMFPush`. For example, `#import <IMFCore/IMFCore.h>` would change to `#import <IMFCompatibility/IMFCompatibility.h>`.
2. Look for a call to the `initializeWithBackendRoute` method and replace the route URL with your on-premises MobileFirst Server URL. For example, replace

   ```
   [[ IMFClient sharedInstance ] initializeWithBackendRoute :@"http://compatibilitydemo.ng.mybluem
   ```

   with

   ```
   [[ IMFClient sharedInstance ] initializeWithBackendRoute :@"http://localhost:10080/customAuthA
   ```

   where `customAuthApp` is the name of your application.

Note: The **`backendGUID`** parameter is ignored.

3. Look for all instantiations of the IMFResourceRequest class and update the request URL with an absolute or relative path to the resource. For example, replace

   ```
   IMFResourceRequest * request = [ IMFResourceRequest requestWithPath :@"http://customAuthApp.ng.my
   [request setHTTPMethod: @"GET"];
   ```

   with the absolute path:

   ```
   IMFResourceRequest * request = [ IMFResourceRequest requestWithPath :@"http://localhost:10080/cus
   ```

   or the relative path:

   ```
   IMFResourceRequest * request = [ IMFResourceRequest requestWithPath : @"/adapters/customAuth/hell
   ```

   Note: The setHTTPMethod is not supported in IBM MobileFirst Platform Foundation for iOS. `IMFResourceRequest:requestWithPath` sets the GET HTTP method by default. If you need to use other HTTP methods, call `IMFResourceRequest:requestWithPath:method`.

Additional steps for migrating a Bluemix application with Push for iOS 8

4. If the existing application is enabled with the Push for iOS 8 service, you must add other libraries to your Objective-C or Swift project. To do this, select your `target` > `BuildPhases` > `Link` binary with `Libraries` and then add the following libraries:
   - `libz.dylib`
   - `libstdc++.6.dylib`
   - `libc++.dylib`
   - `libc.dylib`
   - `sqlcipher.framework`

   Important: If you are using Xcode 7, link `libz.tbd`, `libstdc++.6.tbd`, `libc++.tbd`, and `libc.tbd` instead of the corresponding `.dylib` files.
5. Remove any helper methods from `IMFResponse+IMFPushCategory.h`. Alternatively, you can directly print the responseJson on the response (instead of calling the method on the IMFResponse).
6. In the AppDelegate.m file, add `[[WLPush sharedInstance]setTokenFromClient:deviceToken.description];` to method `-(void)application(UIApplication *)application didRegisterForRemoteNotificationsWithDeviceTokenNSData *)deviceToken {}`.

   Tip: Ensure that the registerDeviceToken method is called and the On_Ready_To_Subscribe window is displayed before calling any of the APIs.

What to do next

Deploy the application. For more information, see "Deploying MobileFirst projects" on page 11-1

Additional migration steps for Cloudant data

Complete these additional steps if your application stores data in a Cloudant database.

About this task

Note: The IMFData API is deprecated. For more information about how to migrate to the Cloudant Sync API, see "Storing mobile data in Cloudant" on page 8-151. If you are already using the Cloudant Sync API, no further migration steps are required.

These instructions only cover migrating your app from Bluemix to IBM MobileFirst Platform Foundation for iOS. If you need to migrate your data between Cloudant databases, see Cloudant replication.

Procedure

1. In the `Podfile` file, replace the pod name `IMFData` with the pod name `IMFDataLocal`. For example, the resulting `Podfile` might look like:

```
source 'https://github.com/CocoaPods/Specs.git'
platform :ios, '7.1'
pod 'IMFCompatibility'
pod 'IMFDataLocal'
```

2. Update the initialization method to point to the location of the data proxy. Change the code that is in your original Bluemix application as follows:

   Change

   Objective-C

```
// Get reference to data managerIMFDataManager
 *manager = [IMFDataManager sharedInstance];
```

   Swift

```
//Get reference to data manager
let manager = IMFDataManager.sharedInstance()
```

   to

   Objective-C

```
// Initialize the IMFDataManager
IMFDataManager *manager = [IMFDataManager initializeWithUrl:CLOUDANT_PROXY_URL];
```

   Swift

```
// Initialize the IMFDataManager
let manager:IMFDataManager = IMFDataManager.initializeWithUrl(CLOUDANT_PROXY_URL)
```

3. Configure additional security setup. Follow these steps for OAuth TAI configuration in the `authenticationConfig.xml` file:
   a. The realm must be either be defined as `cloudant` or must have a `grantScope` of `cloudant` defined.

```
<realm name="cloudant" loginModule="myUserAuthLoginModule">
    <className>com.worklight.core.auth.ext.AdapterAuthenticator</className>
</realm>
```

```
<realm name="myUserAuthRealm" loginModule="myUserAuthLoginModule">
    <className>com.worklight.core.auth.ext.AdapterAuthenticator</className>
    <parameter name="grantScope" value="cloudant"/>
</realm>
```

   Create a `cloudant` realm. For example:

```
<realms>
    <realm loginModule="CustomAuthLoginModule" name="cloudant">
        <className>com.worklight.integration.auth.AdapterAuthenticator</className>
        <parameter name="login-function" value="CustomAuthAdapter.onAuthRequired"/>
        <parameter name="logout-function" value="CustomAuthAdapter.onLogout"/>
    </realm>
    <realm loginModule="CustomAuthLoginModule" name="customIdentityRealm">
        <className>com.worklight.integration.auth.AdapterAuthenticator</className>
  <parameter name="login-function" value="CustomAuthAdapter.onAuthRequired"/>
  <parameter name="logout-function" value="CustomAuthAdapter.onLogout"/>
    </realm>
</realms>

<loginModules>
    <loginModule name="CustomAuthLoginModule">
        <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModules>
```

b. To use the MobileFirst Data Proxy for security protection, you must use the IMFData SDK.

   When you initialize the IMFData SDK, you provide the URL of the MobileFirst Data Proxy.

   The databases that you create with the IMFDataManager API are primed with the proxy URL. You can obtain an OAuth token with the `cloudant` scope through coordination with the MobileFirst native authorization manager. This token is then propagated on all requests to the MobileFirst Data Proxy.

API reference

To develop your iOS applications, refer to the MobileFirst API in Objective-C.

MobileFirst client-side API

This collection of topics contains a description of the application programming interface (API) for use in writing client applications with IBM MobileFirst Platform Foundation for iOS.

You can use MobileFirst client-side API capabilities to improve application development, and MobileFirst server-side API to improve client/server integration and communication between mobile applications and back-end systems.

With the MobileFirst client-side API, your mobile application has access to various MobileFirst features during run time, by using libraries that are bundled into the application. The libraries integrate your mobile application with MobileFirst Server by using predefined communication interfaces. The libraries also provide unified access to native device functionality, which simplifies application development.

The MobileFirst client-side API provide access to MobileFirst functions across multiple device platforms and development approaches. Applications that are built by using web technologies can access MobileFirst 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 native development approach benefit from simplified application security and the integration features of MobileFirst tooling.

MobileFirst client-side API components also provide the following features, which improve application development.

Client to server integration

Client to server integration ensures transparent communication between a mobile application that is built with MobileFirst technology, and MobileFirst Server. MobileFirst 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 MobileFirst Server, or updating the web resources of an 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. MobileFirst Server issues a special key when a new encrypted data store is created.

JSONStore

A JSONStore store is included in IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst 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 MobileFirst Platform Foundation for iOS provides a number of functions for location services. Location services enable you to use Geo and WiFi positions to perform various actions.

Objective-C client-side API for iOS apps

Use this API to develop native app for iOS environment.

You can access MobileFirst services from iOS applications by using this Objective-C client-side API.

Note: To develop native iOS applications, you can also use Apple Swift. This language is compatible with Objective-C. For more information about using MobileFirst API from within the Swift project, see “Configuring a Swift application” on page 8-15.

You can find the description of the API in the following file: Objective-C client-side API for iOS apps.

Objective-C client-side API for migrated Bluemix iOS apps

The Objective-C client-side API for migrated apps from IBM Bluemix provides a compatibility layer that translates the code that is in your original Bluemix app to code that is recognized by IBM MobileFirst Platform Foundation for iOS.

You can find the description of the API in the following file: Objective-C client-side API for migrated Bluemix iOS apps

MobileFirst server-side API

Use the server-side API that IBM MobileFirst Platform Foundation for iOS defines to modify the behavior of the servers that your mobile applications rely on.

MobileFirst 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 MobileFirst 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, MobileFirst 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 MobileFirst Server, and do not stay on the mobile device. If you are running MobileFirst 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 MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst server-side JavaScript API comprises a series of packages.

You can find the description of the API in the following file: JavaScript server-side API.

Java server-side API

The MobileFirst server-side Java API comprises a series of packages.

You can find the description of the API in the following file: Java server-side API.

MobileFirst OAuth TAI API

Use this API when your application uses OAuth TAI (Trust Association Interceptors) security. It contains the WLCredential class, which stores the IBM MobileFirst token information of an authenticated principal.

You can find the description of the API in the following file: OAuth TAI API

REST API Administration Services

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

/otu/1.0/*one-time-url-hash*/runtimes/*runtime-name*/downloads/adapters/*adapter-name*

Example

```
https://www.example.com/worklightadmin/otu/1.0/ffabc301/runtimes/myruntime/downloads/adapters/myad
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

`adapter-name`
The name of the adapter.

`one-time-url-hash`
The one-time-url hash code.

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/myadapter?asy
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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 metadata of the deleted adapter.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-adapter-result
  ok="false"
  productVersion="7.1.0">
  <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 metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/myadapter?l
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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 metadata of the specified adapter.

JSON Example

```
{
  "deployTime" : "2014-04-13T00:18:36.979Z",
  "description" : "My first sample adapter",
  "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters
  "name" : "myadapter",
  "platformVersion" : "6.1.0.00.20131126-0630",
  "procedureDetails" : [
    {
      "audit" : false,
      "connectAs" : "SERVER",
      "description" : "Returns something.",
      "displayName" : "getSomething",
      "name" : "getSomething",
      "securityTest" : "mobileTest",
    },
    ...
  ],
  "productVersion" : "7.1.0",
  "projects" : [
    {
      "name" : "myproject",
    },
    ...
  ],
  "resourceSize" : 1024,
  "urls" : [
    {
      "formParameters" : [
        {
          "defaultValue" : "n/a",
          "javaType" : "java.lang.String",
          "name" : "param",
        },
        ...
      ],
      "headerParameters" : [
        {
          "defaultValue" : "n/a",
          "javaType" : "java.lang.String",
          "name" : "param",
        },
        ...
      ],
      "javaClass" : "com.example.MyRestWrapper",
      "javaMethodName" : "multiplyNumbers",
      "method" : "POST",
      "pathParameters" : [
        {
          "defaultValue" : "n/a",
          "javaType" : "java.lang.String",
          "name" : "param",
        },
        ...
      ],
      "queryParameters" : [
        {
          "defaultValue" : "n/a",
          "javaType" : "java.lang.String",
          "name" : "param",
        },
        ...
      ],
      "securedWithScope" : "wl_antiXSRFRealm wl_anonymousUserRealm",
      "uri" : "/multiply",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapter
  deployTime="2014-04-13T00:18:36.979Z"
  description="My first sample adapter"
  link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/myada
  name="myadapter"
  platformVersion="6.1.0.00.20131126-0630"
  productVersion="7.1.0"
  resourceSize="1024">
  <procedureDetails>
    <procedureDetail
      audit="false"
      connectAs="SERVER"
      description="Returns something."
      displayName="getSomething"
      name="getSomething"
      securityTest="mobileTest"/>
    ...
  </procedureDetails>
  <projects>
    <project name="myproject"/>
    ...
  </projects>
  <urls>
    <url
      javaClass="com.example.MyRestWrapper"
      javaMethodName="multiplyNumbers"
      method="POST"
      securedWithScope="wl_antiXSRFRealm wl_anonymousUserRealm"
      uri="/multiply">
      <formParameters>
        <formParameter
          defaultValue="n/a"
          javaType="java.lang.String"
          name="param"/>
        ...
      </formParameters>
      <headerParameters>
        <headerParameter
          defaultValue="n/a"
          javaType="java.lang.String"
          name="param"/>
        ...
      </headerParameters>
      <pathParameters>
        <pathParameter
          defaultValue="n/a"
          javaType="java.lang.String"
          name="param"/>
        ...
      </pathParameters>
      <queryParameters>
        <queryParameter
          defaultValue="n/a"
          javaType="java.lang.String"
          name="param"/>
        ...
      </queryParameters>
    </url>
    ...
  </urls>
</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 IBM MobileFirst Platform Foundation development tools (Studio) that built the adapter.

procedureDetails
The JavaScript procedures of the adapter.

productVersion
The exact product version.

projects
The projects the adapter belong to.

resourceSize
The size of the adapter resource.

urls
The API documentation of the URLs of the REST API provided by the adapter.

The *adapter procedure details* has the following properties:

audit
`true` if calls to the Javascript procedure are logged in the audit log.

connectAs
The optional type of the connection to the back end. Possible values are: `SERVER` (use the connection policy defined for the adapter) or `END_USER` (use the user's identity for the connection).

description
The optional description of the procedure.

displayName
The optional display name of the procedure.

name
The Javascript name of the adapter procedure.

securityTest
The optional name of the security test used to protect the adapter procedure, as defined in the authenticationConfig.xml file.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

The *url* has the following properties:

formParameters
The form parameters.

headerParameters
The header parameters.

javaClass
The Java class

javaMethodName
The Java method

method
The HTTP method.

pathParameters
The path parameters.

queryParameters
The query parameters.

securedWithScope
The set of realms that secure the REST API.

uri
The URI of the REST API.

The *REST API parameter* has the following properties:

defaultValue
The default value.

javaType
The Java type name.

name
The name of the parameter.

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 database 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters?async=false
```

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.

`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 metadata of the deployed adapter.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-adapter-result
  ok="false"
  productVersion="7.1.0">
  <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

400

No adapter data is provided.

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.

Adapters (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters?bookmark=ABC&
```

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.

bookmark
The bookmark for the page if only a part of the list (a page) should be returned. If a bookmark is specified, the offset parameter is 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: 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 metadata of the deployed adapters.

JSON Example

```
{
  "items" : [
    {
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "description" : "My first sample adapter",
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adap
      "name" : "myadapter",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "procedureDetails" : [
        {
          "audit" : false,
          "connectAs" : "SERVER",
          "description" : "Returns something.",
          "displayName" : "getSomething",
          "name" : "getSomething",
          "securityTest" : "mobileTest",
        },
        ...
      ],
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
      "resourceSize" : 1024,
      "urls" : [
        {
          "formParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "headerParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "javaClass" : "com.example.MyRestWrapper",
          "javaMethodName" : "multiplyNumbers",
          "method" : "POST",
          "pathParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "queryParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "securedWithScope" : "wl_antiXSRFRealm wl_anonymousUserRealm",
          "uri" : "/multiply",
        },
        ...
      ],
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "7.1.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<adapters
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="7.1.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deployTime="2014-04-13T00:18:36.979Z"
      description="My first sample adapter"
      link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters
      name="myadapter"
      platformVersion="6.1.0.00.20131126-0630"
      resourceSize="1024">
      <procedureDetails>
        <procedureDetail
          audit="false"
          connectAs="SERVER"
          description="Returns something."
          displayName="getSomething"
          name="getSomething"
          securityTest="mobileTest"/>
        ...
      </procedureDetails>
      <projects>
        <project name="myproject"/>
        ...
      </projects>
      <urls>
        <url
          javaClass="com.example.MyRestWrapper"
          javaMethodName="multiplyNumbers"
          method="POST"
          securedWithScope="wl_antiXSRFRealm wl_anonymousUserRealm"
          uri="/multiply">
          <formParameters>
            <formParameter
              defaultValue="n/a"
              javaType="java.lang.String"
              name="param"/>
            ...
          </formParameters>
          <headerParameters>
            <headerParameter
              defaultValue="n/a"
              javaType="java.lang.String"
              name="param"/>
            ...
          </headerParameters>
          <pathParameters>
            <pathParameter
              defaultValue="n/a"
              javaType="java.lang.String"
              name="param"/>
            ...
          </pathParameters>
          <queryParameters>
            <queryParameter
              defaultValue="n/a"
              javaType="java.lang.String"
              name="param"/>
            ...
          </queryParameters>
        </url>
        ...
      </urls>
    </item>
    ...
  </items>
</adapters>
```

Response Properties

The response has the following properties:

`items`
The array of adapter metadata

`nextPageBookmark`
The bookmark of the next page if only a page of adapters is returned.

`pageNumber`
The page index if only a page of adapters is returned.

`pageSize`
The page size if only a page of adapters is returned.

`prevPageBookmark`
The bookmark of the previous page 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 IBM MobileFirst Platform Foundation development tools (Studio) that built the adapter.

`procedureDetails`
The JavaScript procedures of the adapter.

`projects`
The projects the adapter belong to.

`resourceSize`
The size of the adapter resource.

`urls`
The API documentation of the URLs of the REST API provided by the adapter.

The *adapter procedure details* has the following properties:

`audit`
`true` if calls to the Javascript procedure are logged in the audit log.

connectAs
: The optional type of the connection to the back end. Possible values are: SERVER (use the connection policy defined for the adapter) or END_USER (use the user's identity for the connection).

description
: The optional description of the procedure.

displayName
: The optional display name of the procedure.

name
: The Javascript name of the adapter procedure.

securityTest
: The optional name of the security test used to protect the adapter procedure, as defined in the authenticationConfig.xml file.

The *project* has the following properties:

name
: The name of the project, which is the context root of the runtime.

The *url* has the following properties:

formParameters
: The form parameters.

headerParameters
: The header parameters.

javaClass
: The Java class

javaMethodName
: The Java method

method
: The HTTP method.

pathParameters
: The path parameters.

queryParameters
: The query parameters.

securedWithScope
: The set of realms that secure the REST API.

uri
: The URI of the REST API.

The *REST API parameter* has the following properties:

defaultValue
: The default value.

javaType
: The Java type name.

name
: The name of the parameter.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime is not found.`

500

`An internal error occurred.`

Adobe Air Application Binary (GET)

Retrieves the Adobe Air 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`**
- **`worklightmonitor`**
- **`worklightoperator`**

Method

```
GET
```

Path

/otu/1.0/*one-time-url-hash*/runtimes/*runtime-name*/downloads/air/*application-name*/*application-version*

Example

```
https://www.example.com/worklightadmin/otu/1.0/ffabc301/runtimes/myruntime/downloads/air/myapplicatio
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

`application-name`
The name of the application.

`application-version`
The application version number.

`one-time-url-hash`
The one-time-url hash code.

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.`

APNS Credentials (DELETE)

Deletes Apple Push Notification Service (APNS) credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

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

Method

`DELETE`

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/apnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The delete status of APNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_PUSH_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteAPNSCredentialsStatus
  status="Success"
  type="DELETE_PUSH_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteAPNSCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

`project`
Project name.

`status`
The APNS credentials are deleted successfully.

`type`
Transaction type.

The *productVersion* has the following properties:

`productVersion`
The exact product version

The *project* has the following properties:

`name`
Name of the project

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.`

APNS Credentials (GET)

Retrieves Apple Push Notification Service (APNS) credentials of the application with the application ID, environment, and version.

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*/apnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The APNS credentials of the application such as certificate information, password, and product version.

JSON Example

```
{
  "certificateExpirationDate" : 2015-05-05T06:29:10.000Z,
  "certificateFileName" : "apns-certificate-sandbox.p12",
  "password" : "password",
  "productVersion" : "7.1.0",
  "sandbox" : true,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<apnsCredentials
  certificateExpirationDate="2015-05-05T06:29:10.000Z"
  certificateFileName="apns-certificate-sandbox.p12"
  password="password"
  productVersion="7.1.0"
  sandbox="true"/>
```

Response Properties

The response has the following properties:

certificateExpirationDate
The expiry date of Certificate.

certificateFileName
The name of the certificate.

password
The password of the certificate.

productVersion
The exact product version.

sandbox
The sandbox is true if certificate is of sandbox type.

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.
```

APNS Credentials (PUT)

Sets Apple Push Notification Service (APNS) credentials of the application with the application ID, environment, version, password, certificate file name, and certificate.

Description

The payload is the form data in which password, certificate file name, and certificate are submitted.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/apnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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

multipart/form-data

Produces

application/json, application/xml, text/xml

Form-data Parameters

certificateFileName
(string) The certificate file name.

password
(string) The certificate password.

certificate
(File) The certificate file.

Response

The status of set APNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_APNS_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<setAPNSCredentialsStatus
  status="Success"
  type="SET_APNS_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</setAPNSCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The APNS credentials are saved successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myapplica
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

JSON Example

```
{
  "action" : "NOTIFY",
  "downloadLink" : "ibmappctr://myapp",
  "message" : "Please update!",
  "multiLanguageMessage" : [
    {
      "locale" : "de",
      "message" : "Bitte updaten!",
    },
    ...
  ],
}
```

XML Example

```
<?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), `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 metadata of the app version and its access rule.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-accessrule-result
  ok="false"
  productVersion="7.1.0">
  <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 (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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myapplicat
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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 metadata of the deleted app version.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-appversion-result
  ok="false"
  productVersion="7.1.0">
  <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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappli
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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

JSON Example

```
{
  "lock" : true,
  "warning" : "true",
}
```

Payload Properties

The payload has the following properties:

lock
Whether the app version is locked.

warning
When a warning happens, provides the details.

Response

JSON Example

```
{
  "ok" : true,
  "warning" : "true",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-appversion-lock-result
  ok="true"
  warning="true"/>
```

Response Properties

The response has the following properties:

ok Whether the operation was successful.

`warning`
When a warning happens, provides the details.

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

/otu/1.0/*one-time-url-hash*/runtimes/*runtime-name*/downloads/applications/*application-name*/*application-env*/*application-version*

Example

```
https://www.example.com/worklightadmin/otu/1.0/ffabc301/runtimes/myruntime/downloads/applications/
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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.

one-time-url-hash
The one-time-url hash code.

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappli
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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 metadata of the deleted application.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-application-result
  ok="false"
  productVersion="7.1.0">
  <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 metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications/myappli
```

Path Parameters

runtime-name
: The name of the runtime. This is the context 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 metadata of the specified application.

JSON Example

```
{
  "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.",
      },
      "authenticityConfig" : "BASIC",
      "buildTime" : "2014-03-29T00:18:36.979Z",
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "deviceProvisioningRealm" : "myProvRealm",
      "envPlatformVersion" : "7.1.0",
      "environment" : "android",
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
      "prevBuildTime" : "2014-03-29T00:18:36.979Z",
      "resourceSize" : 5120,
      "securityTest" : "mobileTest",
      "supportRemoteDisable" : true,
      "supportsAuthenticity" : true,
      "userAuthenticationRealm" : "myAuthRealm",
      "version" : "1.0",
      "versionLocked" : false,
    },
    ...
  ],
  "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applicatio
  "name" : "myapplication",
  "platformVersion" : "6.1.0.00.20131126-0630",
  "productVersion" : "7.1.0",
  "projects" : [
    {
      "name" : "myproject",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<application
  description="My first sample application"
  displayName="My Sample Application"
  link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications
  name="myapplication"
  platformVersion="6.1.0.00.20131126-0630"
  productVersion="7.1.0">
  <environments>
    <environment
      authenticityConfig="BASIC"
      buildTime="2014-03-29T00:18:36.979Z"
      deployTime="2014-04-13T00:18:36.979Z"
      deviceProvisioningRealm="myProvRealm"
      envPlatformVersion="7.1.0"
      environment="android"
      link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applicat
      prevBuildTime="2014-03-29T00:18:36.979Z"
      resourceSize="5120"
      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 IBM MobileFirst Platform Foundation 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.

authenticityConfig
The application authenticity configuration. Possible values are: `NONE`, `BASIC`, `EXTENDED`.

buildTime
The time stamp when the app version was built.

deployTime
The date in ISO 8601 format when the application was deployed.

deviceProvisioningRealm
The name of the realm used for device provisioning.

envPlatformVersion
The version of the platform of the environment.

environment
The platform environment of the app version: android, iphone, ...

link
The URL to access detail information about the application version.

prevBuildTime
The time stamp when the app that was previosuly deployed was built.

publishUrl
For web applications, this is the URL under which the web application was published.

resourceSize
The size of the wlapp file of the application version.

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 authentication.

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 or not running, or the application is not found.
```

500

```
An internal error occurred.
```

Application (POST)

Deploys an application.

Description

It first checks whether the input application is valid. Then, it transfers the application to the database 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications?async=fals
```

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.

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 metadata of the deployed application.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deploy-application-result
  ok="false"
  productVersion="7.1.0">
  <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

400

No application data is provided.

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.

Applications (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applications?bookmar
```

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.

bookmark
The bookmark for the page if only a part of the list (a page) should be returned. If a bookmark is specified, the offset parameter is 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: name. 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 metadata of the deployed applications.

JSON Example

```
{
  "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.",
          },
          "authenticityConfig" : "BASIC",
          "buildTime" : "2014-03-29T00:18:36.979Z",
          "deployTime" : "2014-04-13T00:18:36.979Z",
          "deviceProvisioningRealm" : "myProvRealm",
          "envPlatformVersion" : "7.1.0",
          "environment" : "android",
          "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/app
          "prevBuildTime" : "2014-03-29T00:18:36.979Z",
          "resourceSize" : 5120,
          "securityTest" : "mobileTest",
          "supportRemoteDisable" : true,
          "supportsAuthenticity" : true,
          "userAuthenticationRealm" : "myAuthRealm",
          "version" : "1.0",
          "versionLocked" : false,
        },
        ...
      ],
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
      "name" : "myapplication",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "7.1.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<applications
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="7.1.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      description="My first sample application"
      displayName="My Sample Application"
      link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applicat
      name="myapplication"
      platformVersion="6.1.0.00.20131126-0630">
      <environments>
        <environment
          authenticityConfig="BASIC"
          buildTime="2014-03-29T00:18:36.979Z"
          deployTime="2014-04-13T00:18:36.979Z"
          deviceProvisioningRealm="myProvRealm"
          envPlatformVersion="7.1.0"
          environment="android"
          link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/appl
          prevBuildTime="2014-03-29T00:18:36.979Z"
          resourceSize="5120"
          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 metadata

`nextPageBookmark`
The bookmark of the next page if only a page of applications is returned.

pageNumber
The page index if only a page of applications is returned.

pageSize
The page size if only a page of applications is returned.

prevPageBookmark
The bookmark of the previous page 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 IBM MobileFirst Platform Foundation 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.

authenticityConfig
The application authenticity configuration. Possible values are: `NONE`, `BASIC`, `EXTENDED`.

buildTime
The time stamp when the app version was built.

deployTime
The date in ISO 8601 format when the application was deployed.

deviceProvisioningRealm
The name of the realm used for device provisioning.

envPlatformVersion
The version of the platform of the environment.

environment
The platform environment of the app version: android, iphone, ...

link
The URL to access detail information about the application version.

prevBuildTime
The time stamp when the app that was previosuly deployed was built.

publishUrl
For web applications, this is the URL under which the web application was published.

resourceSize
The size of the wlapp file of the application version.

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 authentication.

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 or not running.
```

500

`An internal error occurred.`

Associate beacons and triggers (DELETE)

Deletes the association of beacons and triggers with the UUID, major number, minor number and triggerName.

Description

Deleting a beacon or beacon-trigger would delete the corresponding beacon-to-trigger associations as well.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/beaconTriggerAssociations/applications/*application-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggerAssociati
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

major
Mandatory. The major number of the beacon whose trigger-association must be deleted.

minor
Mandatory. The minor number of the beacon whose trigger-association must be deleted.

triggerName
Mandatory. The name of the beacon trigger whose beacon-association must be deleted.

uuid
Mandatory. The UUID of the beacon whose trigger-association must be deleted.

Produces

application/json, application/xml, text/xml

Response

The status of the delete of the beacon trigger association.

JSON Example

```
{
  "beaconTriggerAssociations" : {
    "major" : 1,
    "minor" : 4439,
    "triggerName" : "DwellInsideLoanSection",
    "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
  },
  "productVersion" : "7.1.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "errors" : [
      {
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "SUCCESS",
    "timeCreated" : "2014-11-14T05:21:13.404Z",
    "timeUpdated" : "2014-11-14T05:21:13.456Z",
    "type" : "DELETE_BEACON_AND_TRIGGER_ASSOCIATION",
    "userName" : "demouser",
    "warnings" : [
      {
      },
      ...
    ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-beacon-trigger-association-result productVersion="7.1.0">
  <beaconTriggerAssociations
    major="1"
    minor="4439"
    triggerName="DwellInsideLoanSection"
    uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef"/>
  <transaction
    appServerId="Tomcat"
    id="1"
    status="SUCCESS"
    timeCreated="2014-11-14T05:21:13.404Z"
    timeUpdated="2014-11-14T05:21:13.456Z"
    type="DELETE_BEACON_AND_TRIGGER_ASSOCIATION"
    userName="demouser">
    <errors>
      <error/>
      ...
    </errors>
    <project name="myproject"/>
    <warnings>
      <warning/>
      ...
    </warnings>
  </transaction>
</delete-beacon-trigger-association-result>
```

Response Properties

The response has the following properties:

`beaconTriggerAssociations`
The details of the beacon trigger association that is deleted.

`productVersion`
The exact product version.

`transaction`
The details of the transaction.

The *beaconTriggerAssociations* has the following properties:

`major`
The major number of the beacon whose trigger-associations must be deleted.

`minor`
The minor number of the beacon whose trigger-associations must be deleted.

`triggerName`
An unique name for this beacon trigger.

`uuid`
The UUID of beacon whose trigger-associations must be deleted

The *transaction* has the following properties:

`appServerId`
The id of the web application server.

`errors`
Errors, if any, that occurred during the transaction.

`id` The id of the transaction.

`project`
The current project.

`status`
The state of the transaction: `SUCCESS`, `FAILURE`.

`timeCreated`
The date in ISO 8601 format when the transaction was started.

`timeUpdated`
The date in ISO 8601 format when the transaction was completed.

`type`
The type of the transaction: `DELETE_BEACON_AND_TRIGGER_ASSOCIATION`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
Not Found - Either: a) beacon with specified by UUID, major and minor is not found, OR b) trigger
```

500

```
An internal error occurred.
```

Associate beacons and triggers (GET)

Retrieves the association of beacons and triggers with the UUID, major number, minor number, and triggerName.

Description

The beacons and triggers associations are retrieved based on the following query parameters:

- None are specified: Returns all beacon and trigger associations of this application.
- Only triggerName is specified: Returns the associations of the specified trigger with any of the beacons.
- Only UUID is specified with/without triggerName (major and minor number are not specified): Returns the associations of the specified/any trigger with any of the beacons that have matching UUID.
- Only UUID and major number are specified with/without triggerName (minor is not specified): Returns the associations of the specified/any trigger with any of the beacons that have matching UUID and major number.
- Only UUID and minor number are specified with/without triggerName (major is not specified): Returns the associations of the specified/any trigger with any of the beacons that have matching UUID and minor number.
- UUID, major, and minor number are specified with/without triggerName: Returns the associations of the specified/any trigger with the specified beacon.

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*/beaconTriggerAssociations/applications/*application-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggerAssociati
```

Path Parameters

runtime-name
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

application-name
The name of the application.

Query Parameters

Query parameters are optional.

errorIfNotFound
If this flag is set to true (default value) with any of the uuid/major/minor/triggerName parameters specified, and there are no matching beacon trigger associations, then 'HTTP 404 Not Found' error is returned instead of an empty list in the output.

locale
The locale used for error messages.

major
The major number of the beacon whose trigger-associations must be fetched.

minor
The minor number of the beacon whose trigger-associations must be fetched.

triggerName
The name of beacon trigger whose beacon-associations must be fetched.

uuid
The UUID of the beacon whose trigger-associations must be fetched.

Produces

application/json, application/xml, text/xml

Response

The details of all the beacon trigger associations that are retrieved.

JSON Example

```
{
  "beaconTriggerAssociations" : [
    {
      "major" : 1,
      "minor" : 4439,
      "triggerName" : "DwellInsideLoanSection",
      "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
    },
    ...
  ],
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<list-beacon-trigger-associations-result productVersion="7.1.0">
  <beaconTriggerAssociations>
    <beaconTriggerAssociation
      major="1"
      minor="4439"
      triggerName="DwellInsideLoanSection"
      uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef"/>
    ...
  </beaconTriggerAssociations>
</list-beacon-trigger-associations-result>
```

Response Properties

The response has the following properties:

beaconTriggerAssociations
The array of beacon trigger associations.

productVersion
The exact product version.

The *beaconTriggerAssociations* has the following properties:

major
The major number of the beacon.

minor
The minor number of the beacon.

triggerName
The unique name of the beacon trigger.

uuid
The UUID of the beacon.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
The errorIfNotFound flag is set to true (or not specified) and one of the following conditions hap
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json, app
```

500

```
An internal error occurred.
```

Associate beacons and triggers (PUT)

Associates the specified beacons with the specified triggers.

Description

Use this API to specify a trigger and the list of beacons to associate with it. Or to specify a beacon and the list of triggers to associate with it.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/beaconTriggerAssociations/applications/*application-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggerAssociati
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

The payload has values for the beacons and the trigger names. It can be in JSON or XML format.

JSON Example

```
{
  "beacons" : [
    {
      "major" : 1,
      "minor" : 4439,
      "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
    },
    ...
  ],
  "triggers" : [
    {
      "triggerName" : "DwellInsideLoanSection",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beaconTriggerAssociations>
  <beacons>
    <beacon
      major="1"
      minor="4439"
      uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef"/>
    ...
  </beacons>
  <triggers>
    <trigger triggerName="DwellInsideLoanSection"/>
    ...
  </triggers>
</beaconTriggerAssociations>
```

Payload Properties

The payload has the following properties:

beacons
List of beacons to which each of the listed triggers must be associated with. Each beacon is identified by its UUID, major and minor number.

triggers
List of triggers to which each of the listed beacons must be associated with. Each beacon-trigger is identified by its triggerName.

The *beaconTriggers* has the following properties:

triggerName
The name of beacon-trigger.

The *beacons* has the following properties:

major
The major number of the beacon.

minor
The minor number of the beacon.

uuid
The UUID of the beacon.

Response

The status of the association of beacons and triggers.

JSON Example

```
{
  "beaconTriggerAssociations" : [
    {
      "major" : 1,
      "minor" : 4439,
      "triggerName" : "DwellInsideLoanSection",
      "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
    },
    ...
  ],
  "productVersion" : "7.1.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "errors" : [
      {
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "SUCCESS",
    "timeCreated" : "2014-11-14T05:21:13.404Z",
    "timeUpdated" : "2014-11-14T05:21:13.456Z",
    "type" : "ASSOCIATE_BEACONS_AND_TRIGGERS",
    "userName" : "demouser",
    "warnings" : [
      {
      },
      ...
    ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<add-beacon-trigger-associations-result productVersion="7.1.0">
  <beaconTriggerAssociations>
    <beaconTriggerAssociation
      major="1"
      minor="4439"
      triggerName="DwellInsideLoanSection"
      uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef"/>
    ...
  </beaconTriggerAssociations>
  <transaction
    appServerId="Tomcat"
    id="1"
    status="SUCCESS"
    timeCreated="2014-11-14T05:21:13.404Z"
    timeUpdated="2014-11-14T05:21:13.456Z"
    type="ASSOCIATE_BEACONS_AND_TRIGGERS"
    userName="demouser">
    <errors>
      <error/>
      ...
    </errors>
    <project name="myproject"/>
    <warnings>
```

```
      <warning/>
      ...
    </warnings>
  </transaction>
</add-beacon-trigger-associations-result>
```

Response Properties

The response has the following properties:

beaconTriggerAssociations
The list of the beacon trigger associations that are created.

productVersion
The exact product version.

transaction
The details of the transaction.

The *beaconTriggerAssociations* has the following properties:

major
The major number of the beacon that is associated with a specified beacon-trigger.

minor
The minor number of the beacon that is associated with a specified beacon-trigger.

triggerName
An unique name for this beacon trigger.

uuid
The UUID of the beacon that is associated with a specified beacon-trigger.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: `SUCCESS`, `FAILURE`.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: `ASSOCIATE_BEACONS_AND_TRIGGERS`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

Bad Request - The request could not be understood by the server due to malformed syntax.

401

Unauthorized - The caller is either not authenticated or not authorized to make this request.

404

Not Found - One/more of the specified beacons or beacon-triggers not found.

406

Unsupported Accept type - The content type specified in Accept header is not application/json, appli

415

Unsupported Media Type - The server is refusing to service the request because the request payload i

500

An internal error occurred.

Beacon Trigger (DELETE)

Deletes the beacon trigger by using the triggerName.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

DELETE

Path

/management-apis/1.0/runtimes/*runtime-name*/beaconTriggers/*trigger-name*

Example

https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggers/mytrigg

Path Parameters

runtime-name
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

trigger-name
The name of the beacon trigger.

Query Parameters

Query parameters are optional.

`locale`
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The status of the delete of the beacon trigger.

JSON Example

```
{
  "beaconTrigger" : {
    "actionPayload" : {
      "alert" : "Avail lowest interest rate of just 7.5% on home loans!",
    },
    "dwellingTime" : 5000,
    "proximityState" : "Far",
    "triggerName" : "DwellInsideLoanSection",
    "triggerType" : "Enter",
  },
  "productVersion" : "7.1.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "errors" : [
      {
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "SUCCESS",
    "timeCreated" : "2014-11-14T05:21:13.404Z",
    "timeUpdated" : "2014-11-14T05:21:13.456Z",
    "type" : "DELETE_BEACON_TRIGGER",
    "userName" : "demouser",
    "warnings" : [
      {
      },
      ...
    ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<delete-beacon-trigger-result productVersion="7.1.0">
  <beaconTrigger
    dwellingTime="5000"
    proximityState="Far"
    triggerName="DwellInsideLoanSection"
    triggerType="Enter">
    <actionPayload alert="Avail lowest interest rate of just 7.5% on home loans!"/>
  </beaconTrigger>
  <transaction
    appServerId="Tomcat"
    id="1"
    status="SUCCESS"
    timeCreated="2014-11-14T05:21:13.404Z"
    timeUpdated="2014-11-14T05:21:13.456Z"
    type="DELETE_BEACON_TRIGGER"
    userName="demouser">
    <errors>
      <error/>
      ...
    </errors>
    <project name="myproject"/>
    <warnings>
      <warning/>
      ...
    </warnings>
  </transaction>
</delete-beacon-trigger-result>
```

Response Properties

The response has the following properties:

beaconTrigger
The details of the beacon trigger that is deleted.

productVersion
The exact product version.

transaction
The details of the transaction.

The *beaconTrigger* has the following properties:

actionPayload
The details of an action to be taken when the trigger is activated.

dwellingTime
Optional: Applicable only for triggerTypes: DwellInside and DwellOutside. It should be specified in milliseconds and defines how long the device must be inside, or outside a beacon region before the dwellInside or dwellOutside trigger is activated. Mandatory with triggerType of DwellInside and DwellOutside.

proximityState
Optional: The proximity state that is specified for a beacon trigger. It can be either Immediate, Near, or Far. The default value is Far.

triggerName
An unique name for this beacon trigger.

triggerType
The type of beacon trigger. The value of the type can be either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message to be sent when a trigger is activated.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: SUCCESS, FAILURE.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: DELETE_BEACON_TRIGGER.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
Not Found - A beacon trigger with the specified triggerName is not found.
```

500

```
An internal error occurred.
```

Beacon Trigger (GET)

Retrieves the beacon trigger with the triggerName.

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*/beaconTriggers/*trigger-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggers/mytrigge
```

Path Parameters

runtime-name
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

trigger-name
The name of the beacon trigger.

Query Parameters

Query parameters are optional.

locale
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The details of the beacon trigger that is retrieved.

JSON Example

```
{
  "beaconTrigger" : {
    "actionPayload" : {
      "alert" : "Avail lowest interest rate of just 7.5% on home loans.",
    },
    "dwellingTime" : 5000,
    "proximityState" : "Near",
    "triggerName" : "DwellInsideLoanSection",
    "triggerType" : "Enter",
  },
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<show-beacon-trigger-result productVersion="7.1.0">
  <beaconTrigger
    dwellingTime="5000"
    proximityState="Near"
    triggerName="DwellInsideLoanSection"
    triggerType="Enter">
    <actionPayload alert="Avail lowest interest rate of just 7.5% on home loans."/>
  </beaconTrigger>
</show-beacon-trigger-result>
```

Response Properties

The response has the following properties:

beaconTrigger
The beacon trigger that was found.

productVersion
The exact product version.

The *beaconTrigger* has the following properties:

actionPayload
The details of the action that is taken when the trigger is activated.

dwellingTime
Available only for triggerTypes: DwellInside and DwellOutside. It is the time in milliseconds that specifies how long the device must be inside, or outside the associated beacon region before the DwellInside or DwellOutside trigger is activated.

proximityState
The proximity state that was specified for the beacon trigger. It is either Immediate, Near, or Far.

triggerName
The unique name of the beacon trigger.

triggerType
The type of beacon trigger. It is either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message that is shown on the mobile device of user when this trigger is activated.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`Not Found - No beacon-trigger found with matching triggerName.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json, app`

500

`An internal error occurred.`

Beacon Triggers (GET)

Retrieves all the beacon triggers.

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*/beaconTriggers

Example

`https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggers?locale=`

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.

Produces

application/json, application/xml, text/xml

Response

The details of all the beacon triggers that are retrieved.

JSON Example

```
{
  "beaconTriggers" : [
    {
      "actionPayload" : {
        "alert" : "Avail lowest interest rate of just 7.5% on home loans.",
      },
      "dwellingTime" : 5000,
      "proximityState" : "Near",
      "triggerName" : "DwellInsideLoanSection",
      "triggerType" : "Enter",
    },
    ...
  ],
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<list-beacon-triggers-result productVersion="7.1.0">
  <beaconTriggers>
    <beaconTrigger
      dwellingTime="5000"
      proximityState="Near"
      triggerName="DwellInsideLoanSection"
      triggerType="Enter">
      <actionPayload alert="Avail lowest interest rate of just 7.5% on home loans."/>
    </beaconTrigger>
    ...
  </beaconTriggers>
</list-beacon-triggers-result>
```

Response Properties

The response has the following properties:

beaconTriggers
The array of the beacon triggers.

productVersion
The exact product version.

The *beaconTriggers* has the following properties:

actionPayload
The details of the action that is taken when the trigger is activated.

dwellingTime
Available only for triggerTypes: DwellInside and DwellOutside. It is the time in milliseconds that specifies how long the device must be inside, or outside the associated beacon region before the DwellInside or DwellOutside trigger is activated.

proximityState
The proximity state that was specified for the beacon trigger. It is either Immediate, Near, or Far.

triggerName
The unique name of the beacon trigger.

triggerType
The type of beacon trigger. It is either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message that is shown on the mobile device of user when this trigger is activated.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json, app
```

500

```
An internal error occurred.
```

Beacon Triggers (POST)

Adds a new beacon trigger by using the triggerName, triggerType, proximityState, and actionPayload properties.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

POST

Path

/management-apis/1.0/runtimes/*runtime-name*/beaconTriggers

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggers?locale=
```

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.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

The payload has values for the triggerName, triggerType, proximityState, dwellingTime, and actionPayload properties. It can be in JSON or XML format

JSON Example

```
{
  "actionPayload" : {
    "alert" : "Avail lowest interest rate of just 7.5% on home loans!",
  },
  "dwellingTime" : 5000,
  "proximityState" : "Far",
  "triggerName" : "DwellInsideLoanSection",
  "triggerType" : "Enter",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beaconTrigger
  dwellingTime="5000"
  proximityState="Far"
  triggerName="DwellInsideLoanSection"
  triggerType="Enter">
  <actionPayload alert="Avail lowest interest rate of just 7.5% on home loans!"/>
</beaconTrigger>
```

Payload Properties

The payload has the following properties:

actionPayload
The details of an action to be taken when the trigger is activated.

dwellingTime
Optional: Applicable only for triggerTypes: DwellInside and DwellOutside. It should be specified in milliseconds and defines how long the device must be inside, or outside a beacon region before the dwellInside or dwellOutside trigger is activated. Mandatory with triggerType of DwellInside and DwellOutside.

proximityState
Optional: The proximity state that is specified for a beacon trigger. It can be either Immediate, Near, or Far. The default value is Far.

triggerName
An unique name for this beacon trigger (consisting of alphanumeric or underscore characters and beginning with an alphabet).

triggerType
The type of beacon trigger. The value of the type can be either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message to be sent when a trigger is activated.

Response

The status of the adding of the beacon trigger.

JSON Example

```
{
  "beaconTrigger" : {
    "actionPayload" : {
      "alert" : "Avail lowest interest rate of just 7.5% on home loans!",
    },
    "dwellingTime" : 5000,
    "proximityState" : "Far",
    "triggerName" : "DwellInsideLoanSection",
    "triggerType" : "Enter",
  },
  "productVersion" : "7.1.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "errors" : [
      {
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "SUCCESS",
    "timeCreated" : "2014-11-14T05:21:13.404Z",
    "timeUpdated" : "2014-11-14T05:21:13.456Z",
    "type" : "ADD_BEACON_TRIGGER",
    "userName" : "demouser",
    "warnings" : [
      {
      },
      ...
    ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-beacon-trigger-result productVersion="7.1.0">
  <beaconTrigger
    dwellingTime="5000"
    proximityState="Far"
    triggerName="DwellInsideLoanSection"
    triggerType="Enter">
    <actionPayload alert="Avail lowest interest rate of just 7.5% on home loans!"/>
  </beaconTrigger>
  <transaction
    appServerId="Tomcat"
    id="1"
    status="SUCCESS"
    timeCreated="2014-11-14T05:21:13.404Z"
    timeUpdated="2014-11-14T05:21:13.456Z"
    type="ADD_BEACON_TRIGGER"
    userName="demouser">
    <errors>
      <error/>
      ...
    </errors>
    <project name="myproject"/>
    <warnings>
      <warning/>
      ...
    </warnings>
  </transaction>
</set-beacon-trigger-result>
```

Response Properties

The response has the following properties:

beaconTrigger
The details of the beacon trigger that is created.

productVersion
The exact product version.

transaction
The details of the transaction.

The *beaconTrigger* has the following properties:

actionPayload
The details of an action to be taken when the trigger is activated.

dwellingTime
Optional: Applicable only for triggerTypes: DwellInside and DwellOutside. It should be specified in milliseconds and defines how long the device must be inside, or outside a beacon region before the dwellInside or dwellOutside trigger is activated. Mandatory with triggerType of DwellInside and DwellOutside.

proximityState
Optional: The proximity state that is specified for a beacon trigger. It can be either Immediate, Near, or Far. The default value is Far.

triggerName
An unique name for this beacon trigger.

triggerType
The type of beacon trigger. The value of the type can be either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message to be sent when a trigger is activated.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: `SUCCESS`, `FAILURE`.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: `ADD_BEACON_TRIGGER`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

`Bad Request - The request could not be understood by the server due to malformed syntax.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json, appli`

409

`Conflict - There is already an existing trigger with the specified triggerName.`

415

`Unsupported Media Type - The server is refusing to service the request because the request payload i`

500

`An internal error occurred.`

Beacon Triggers (PUT)

Updates the beacon trigger that is specified by using the triggerName property. Other properties (triggerType, proximityState, dwellingTime, and actionPayload) are optional. Only those that need to be updated must be specified.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/beaconTriggers

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/beaconTriggers?locale=
```

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.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

The payload has values for the triggerName, triggerType, proximityState, dwellingTime, and actionPayload properties. It can be in JSON or XML format.

JSON Example

```
{
  "actionPayload" : {
    "alert" : "Avail lowest interest rate of just 7.25% on home loans!",
  },
  "triggerName" : "DwellInsideLoanSection",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beaconTrigger triggerName="DwellInsideLoanSection">
  <actionPayload alert="Avail lowest interest rate of just 7.25% on home loans!"/>
</beaconTrigger>
```

Payload Properties

The payload has the following properties:

actionPayload
The details of an action to be taken when the trigger is activated.

triggerName
An unique name for this beacon trigger.

The *actionPayload* has the following properties:

alert
The alert message to be sent when a trigger is activated.

Response

The status of the update of the beacon trigger.

JSON Example

```
{
  "beaconTrigger" : {
    "actionPayload" : {
      "alert" : "Avail lowest interest rate of just 7.25% on home loans!",
    },
    "dwellingTime" : 5000,
    "proximityState" : "Far",
    "triggerName" : "DwellInsideLoanSection",
    "triggerType" : "Enter",
  },
  "productVersion" : "7.1.0",
  "transaction" : {
    "appServerId" : "Tomcat",
    "errors" : [
      {
      },
      ...
    ],
    "id" : 1,
    "project" : {
      "name" : "myproject",
    },
    "status" : "SUCCESS",
    "timeCreated" : "2014-11-14T05:21:13.404Z",
    "timeUpdated" : "2014-11-14T05:21:13.456Z",
    "type" : "UPDATE_BEACON_TRIGGER",
    "userName" : "demouser",
    "warnings" : [
      {
      },
      ...
    ],
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-beacon-trigger-result productVersion="7.1.0">
  <beaconTrigger
    dwellingTime="5000"
    proximityState="Far"
    triggerName="DwellInsideLoanSection"
    triggerType="Enter">
    <actionPayload alert="Avail lowest interest rate of just 7.25% on home loans!"/>
  </beaconTrigger>
  <transaction
    appServerId="Tomcat"
    id="1"
    status="SUCCESS"
    timeCreated="2014-11-14T05:21:13.404Z"
    timeUpdated="2014-11-14T05:21:13.456Z"
    type="UPDATE_BEACON_TRIGGER"
    userName="demouser">
    <errors>
      <error/>
      ...
    </errors>
    <project name="myproject"/>
    <warnings>
      <warning/>
      ...
    </warnings>
  </transaction>
</set-beacon-trigger-result>
```

Response Properties

The response has the following properties:

beaconTrigger
The details of the beacon trigger that is updated.

productVersion
The exact product version.

transaction
The details of the transaction.

The *beaconTrigger* has the following properties:

actionPayload
The details of an action to be taken when the trigger is activated.

dwellingTime
Optional: Applicable only for triggerTypes: DwellInside and DwellOutside. It should be specified in milliseconds and defines how long the device must be inside, or outside a beacon region before the dwellInside or dwellOutside trigger is activated. Mandatory with triggerType of DwellInside and DwellOutside.

proximityState
Optional: The proximity state that is specified for a beacon trigger. It can be either Immediate, Near, or Far. The default value is Far.

triggerName
An unique name for this beacon trigger.

triggerType
The type of beacon trigger. The value of the type can be either Enter, Exit, DwellInside, or DwellOutside.

The *actionPayload* has the following properties:

alert
The alert message to be sent when a trigger is activated.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: `SUCCESS`, `FAILURE`.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: `UPDATE_BEACON_TRIGGER`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

```
Bad Request - The request could not be understood by the server due to malformed syntax.
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
Not Found - A beacon-trigger with specified triggerName does not exist.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json, appli
```

415

```
Unsupported Media Type - The server is refusing to service the request because the request payload i
```

500

```
An internal error occurred.
```

Beacons (DELETE)

Deletes the beacon by using the UUID, the major number, and minor number.

Description

Each of these query parameters (uuid, major and minor) are mandatory. If any of them are missing, the request fails with 400 Bad Request.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
DELETE
```

Path

/management-apis/1.0/beacons

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/beacons?locale=de_DE&major=1&minor=4439&u
```

Query Parameters

Query parameters are optional.

locale
The locale used for error messages.

major
Mandatory. The major number of the beacon.

minor
Mandatory. The minor number of the beacon.

uuid
Mandatory. The UUID of the beacon.

Produces

application/json, application/xml, text/xml

Response

The status of the delete of the beacon. The transaction details might be empty if there are no runtimes deployed.

JSON Example

```
{
  "beacon" : {
    "major" : 1,
    "minor" : 4439,
    "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
  },
  "ok" : false,
  "productVersion" : "7.1.0",
  "transactions" : [
    {
      "appServerId" : "Tomcat",
      "errors" : [
        {
        },
        ...
      ],
      "id" : 1,
      "project" : {
        "name" : "myproject",
      },
      "status" : "SUCCESS",
      "timeCreated" : "2014-11-14T05:21:13.404Z",
      "timeUpdated" : "2014-11-14T05:21:13.456Z",
      "type" : "DELETE_BEACON",
      "userName" : "demouser",
      "warnings" : [
        {
        },
        ...
      ],
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<remove-beacon-result
  ok="false"
  productVersion="7.1.0">
  <beacon
```

```
    major="1"
    minor="4439"
    uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef"/>
  <transactions>
    <transaction
      appServerId="Tomcat"
      id="1"
      status="SUCCESS"
      timeCreated="2014-11-14T05:21:13.404Z"
      timeUpdated="2014-11-14T05:21:13.456Z"
      type="DELETE_BEACON"
      userName="demouser">
      <errors>
        <error/>
        ...
      </errors>
      <project name="myproject"/>
      <warnings>
        <warning/>
        ...
      </warnings>
    </transaction>
    ...
  </transactions>
</remove-beacon-result>
```

Response Properties

The response has the following properties:

beacon
The details of the beacon that is deleted.

ok Whether all transactions were successful.

productVersion
The exact product version.

transactions
The details of the transactions, one for each runtime.

The *beacon* has the following properties:

major
The major number of the beacon.

minor
The minor number of the beacon.

uuid
The UUID of the beacon.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: `SUCCESS`, `FAILURE`.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: `DELETE_BEACON`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

```
Bad Request - The request could not be understood by the server due to missing mandatory parameter
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
A beacon with the specified uuid+major+minor numbers is not found.
```

500

```
An internal error occurred.
```

Beacons (GET)

Retrieves the beacon with the UUID, major number, and minor number.

Description

The beacons are retrieved based on which of the query parameters are mentioned:

- UUID, major number, and minor number are all specified: returns the details of a specific beacon.
- Only UUID and major number are specified: returns the details of all beacons with matching UUID and major number.
- Only UUID and minor number are specified: returns the details of all beacons with matching UUID and minor number.
- Only UUID is specified: returns the details of all beacons with matching UUID.
- None are specified: returns the details of all beacons.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**

- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

GET

Path

/management-apis/1.0/beacons

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/beacons?errorIfNotFound=true&locale=de_DE
```

Query Parameters

Query parameters are optional.

errorIfNotFound
If this flag is set to true (default value), and uuid and/or major/minor parameters are specified for which there are no matching beacons, then 'HTTP 404 Not Found' error is returned instead of an empty list in the output.

locale
The locale used for error messages.

major
The major number of the beacon.

minor
The minor number of the beacon.

uuid
The UUID of the beacon.

Produces

application/json, application/xml, text/xml

Response

The details of all the beacons that are retrieved.

JSON Example

```
{
  "beacons" : [
    {
      "customData" : {
        "beaconLocation" : "loanSection",
        "branchName" : "Indiranagar, Bangalore",
      },
      "latitude" : 12.952,
      "longitude" : 77.644,
      "major" : 1,
      "minor" : 4439,
      "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
    },
    ...
  ],
  "productVersion" : 7.1.0,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<list-beacons-result productVersion="7.1.0">
  <beacons>
    <beacon
      latitude="12.952"
      longitude="77.644"
      major="1"
      minor="4439"
      uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef">
      <customData
        beaconLocation="loanSection"
        branchName="Indiranagar, Bangalore"/>
    </beacon>
    ...
  </beacons>
</list-beacons-result>
```

Response Properties

The response has the following properties:

`beacons`
The array of beacons

`productVersion`
The exact product version.

The *beacons* has the following properties:

`customData`
Optional: Any other customer-specific data that is associated with this beacon like branch/store where this beacon is deployed.

`latitude`
Optional latitude where the beacon is deployed.

`longitude`
Optional longitude where the beacon is deployed.

`major`
The major number of the beacon.

`minor`
The minor number of the beacon.

`uuid`
The UUID of the beacon.

The *customData* has the following properties:

`beaconLocation`
The physical location of the beacon.

`branchName`
The branch where the beacon is installed.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A beacon with the specified uuid and/or major/minor is not found and errorIfNotFound flag was either`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json, appli`

500

`An internal error occurred.`

Beacons (PUT)

Registers (Adds/Updates) the beacon that is identified by UUID, major number, and minor number in the payload.

Roles

Users in the following roles are authorized to perform this operation:

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

Method

`PUT`

Path

/management-apis/1.0/beacons

Example

`https://www.example.com/worklightadmin/management-apis/1.0/beacons?locale=de_DE`

Query Parameters

Query parameters are optional.

`locale`
The locale used for error messages.

Consumes

application/json, application/xml, text/xml

Produces

application/json, application/xml, text/xml

Payload

The payload can be in JSON or XML format and has values for UUID, major number, minor number, latitude, longitude, and customData properties.

JSON Example

```
{
  "customData" : {
    "beaconLocation" : "loanSection",
    "branchName" : "Indiranagar, Bangalore",
  },
  "latitude" : "12.95213",
  "longitude" : 77.64482,
  "major" : 1,
  "minor" : 4439,
  "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<beacon
  latitude="12.95213"
  longitude="77.64482"
  major="1"
  minor="4439"
  uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef">
  <customData
    beaconLocation="loanSection"
    branchName="Indiranagar, Bangalore"/>
</beacon>
```

Payload Properties

The payload has the following properties:

customData
Optional: Any other customer-specific data that is associated with this beacon like branch/store where this beacon is deployed.

latitude
Optional latitude where the beacon is deployed.

longitude
Optional longitude where the beacon is deployed.

major
The major number of the beacon (positive number in the range 0-65535 inclusive).

minor
The minor number of the beacon (positive number in the range 0-65535 inclusive).

uuid
UUID of beacon. UUID must be specified in canonical form and has 32 hexadecimal digits. The digits are specified in five groups and separated by hyphens, in the form 8-4-4-4-12 for a total of 36 characters (32 alphanumeric characters and 4 hyphens).

The *customData* has the following properties:

beaconLocation

The physical location of the beacon.

branchName

The branch where the beacon is installed.

Response

The status of the add/update of the beacon. The transaction details might by empty if there are no runtimes deployed.

JSON Example

```
{
  "beacon" : {
    "customData" : {
      "beaconLocation" : "loanSection",
      "branchName" : "Indiranagar, Bangalore",
    },
    "latitude" : "12.952",
    "longitude" : 77.644,
    "major" : 1,
    "minor" : 4439,
    "uuid" : "3d402cf0-3691-4bd9-97ff-0b0a93a160ef",
  },
  "ok" : true,
  "productVersion" : "7.1.0",
  "transactions" : [
    {
      "appServerId" : "Tomcat",
      "errors" : [
        {
        },
        ...
      ],
      "id" : 1,
      "project" : {
        "name" : "myproject",
      },
      "status" : "SUCCESS",
      "timeCreated" : "2014-11-14T05:21:13.404Z",
      "timeUpdated" : "2014-11-14T05:21:13.456Z",
      "type" : "REGISTER_BEACON",
      "userName" : "demouser",
      "warnings" : [
        {
        },
        ...
      ],
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-beacon-result
  ok="true"
  productVersion="7.1.0">
  <beacon
    latitude="12.952"
    longitude="77.644"
    major="1"
    minor="4439"
    uuid="3d402cf0-3691-4bd9-97ff-0b0a93a160ef">
    <customData
      beaconLocation="loanSection"
      branchName="Indiranagar, Bangalore"/>
  </beacon>
  <transactions>
    <transaction
      appServerId="Tomcat"
      id="1"
      status="SUCCESS"
      timeCreated="2014-11-14T05:21:13.404Z"
      timeUpdated="2014-11-14T05:21:13.456Z"
      type="REGISTER_BEACON"
      userName="demouser">
      <errors>
        <error/>
        ...
      </errors>
      <project name="myproject"/>
      <warnings>
        <warning/>
        ...
      </warnings>
    </transaction>
    ...
  </transactions>
</set-beacon-result>
```

Response Properties

The response has the following properties:

beacon
The details of the beacon that is added/updated.

ok Whether all transactions were successful.

productVersion
The exact product version.

transactions
The details of the transactions, one for each runtime.

The *beacon* has the following properties:

customData
Optional: Any other customer-specific data that is associated with this beacon like branch/store where this beacon is deployed.

latitude
Optional latitude where the beacon is deployed.

longitude
Optional longitude where the beacon is deployed.

major
The major number of the beacon.

minor
The minor number of the beacon.

uuid
The UUID of the beacon.

The *customData* has the following properties:

beaconLocation
The physical location of the beacon.

branchName
The branch where the beacon is installed.

The *transaction* has the following properties:

appServerId
The id of the web application server.

errors
Errors, if any, that occurred during the transaction.

id The id of the transaction.

project
The current project.

status
The state of the transaction: `SUCCESS`, `FAILURE`.

timeCreated
The date in ISO 8601 format when the transaction was started.

timeUpdated
The date in ISO 8601 format when the transaction was completed.

type
The type of the transaction: `REGISTER_BEACON`.

userName
The user that initiated the transaction.

warnings
Warnings, if any, that occurred during the transaction.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

Errors

400

```
Bad Request - The request could not be understood by the server due to malformed syntax or missing ma
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json, appli
```

415

```
Unsupported Media Type - The server is refusing to service the request because the request payload is
```

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`**
- **`worklightdeployer`**
- **`worklightoperator`**

Method

`PUT`

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*/applications/*application-name*

Example

`https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789/a`

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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

JSON Example

```
{
  "status" : "ENABLED",
}
```

Payload Properties

The payload has the following properties:

status
The status of the application: `ENABLED` or `DISABLED`.

Response

The metadata of the transaction.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-applicationdevice-status-result
  ok="false"
  productVersion="7.1.0">
  <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 metadata 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**
- **worklightdeployer**
- **worklightoperator**

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789?asy
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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 metadata of the deleted device.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<remove-device-result
  ok="false"
  productVersion="7.1.0">
  <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 MobileFirst 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**
- **worklightdeployer**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/devices/*device-id*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices/12345-6789?a
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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

JSON Example

```
{
  "status" : "LOST",
}
```

Payload Properties

The payload has the following properties:

status
The new status of the device: `ACTIVE`, `LOST`, `STOLEN`, `EXPIRED`, `DISABLED`.

Response

The metadata of the transaction.

JSON Example

```
{
  "ok" : false,
  "productVersion" : "7.1.0",
  "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",
  },
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<set-device-status-result
  ok="false"
  productVersion="7.1.0">
  <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 metadata for the list of devices that accessed this project.

Note

Since 7.1, the `offset` parameter is no longer supported. Use the bookmark parameter instead for paging.

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/devices?bookmark=ABC
```

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.

bookmark
The bookmark for the page if only a part of the list (a page) should be returned.

locale
The locale used for error messages.

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 metadata of the devices that accessed this project.

JSON Example

```
{
  "items" : [
    {
      "applicationDeviceAssociations" : [
        {
          "appName" : "myapplication",
          "deviceId" : "12345-6789",
          "deviceStatus" : "LOST",
          "status" : "ENABLED",
        },
        ...
      ],
      "deviceEnvironment" : "android",
      "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",
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "7.1.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<devices
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="7.1.0"
  startIndex="0"
  totalListSize="33">
  <items>
    <item
      deviceEnvironment="android"
      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 metadata

`nextPageBookmark`
The bookmark of the next page if only a page of devices is returned.

`pageNumber`
The page index if only a page of devices is returned.

`pageSize`
The page size if only a page of devices is returned.

`prevPageBookmark`
The bookmark of the previous or first page 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: android, iphone, ...

`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 metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/eventsour
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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 metadata of the event source.

JSON Example

```
{
  "numberOfMessagesSent" : 1,
  "numberOfSubscribedUsers" : 1,
  "productVersion" : "7.1.0",
  "qname" : "SampleAdapter.SampleEventSource",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<eventsources
  numberOfMessagesSent="1"
  numberOfSubscribedUsers="1"
  productVersion="7.1.0"
  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 or not running.
```

500

```
An internal error occurred.
```

Event Sources (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/eventsou
```

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.

Produces

application/json, application/xml, text/xml

Response

The metadata of the event sources.

JSON Example

```
{
  "eventsources" : [
    {
      "numberOfMessagesSent" : 1,
      "numberOfSubscribedUsers" : 1,
      "qname" : "myadapter.myeventsource",
    },
    ...
  ],
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<eventsources productVersion="7.1.0">
  <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 metadata

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 or not running.`

500

`An internal error occurred.`

Farm topology members (GET)

Retrieves the list of members of the farm.

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*/farm

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/farm?locale=de_DE
```

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.

Produces

application/json, application/xml, text/xml

Response

The list of nodes registered in the current farm topology

JSON Example

```
{
  "nodes" : [
    {
      "adminUser" : "johndoe",
      "appServerType" : "LIBERTY",
      "heartbeatTime" : "2014-12-08T23:32:04.700Z",
      "host" : "192.168.0.4",
      "pk" : {
        "projectName" : "mytestproject",
        "serverId" : "Farm_Node_3",
      },
      "port" : "8686",
      "status" : "ALIVE",
      "tomcatPort" : "8989",
    },
    ...
  ],
  "numberOfNodes" : 3,
  "productVersion" : "7.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<farm-members
  numberOfNodes="3"
  productVersion="7.0">
  <nodes>
    <node
      adminUser="johndoe"
      appServerType="LIBERTY"
      heartbeatTime="2014-12-08T23:32:04.700Z"
      host="192.168.0.4"
      port="8686"
      status="ALIVE"
      tomcatPort="8989">
      <pk
        projectName="mytestproject"
        serverId="Farm_Node_3"/>
    </node>
    ...
  </nodes>
</farm-members>
```

Response Properties

The response has the following properties:

nodes
The array of farm nodes

numberOfNodes
The total number of nodes.

productVersion
The exact product version.

The *farm node* has the following properties:

adminUser
The user id to use for REST

appServerType
The server type of this node

heartbeatTime
The last heartbeat time

host
The hostname of this node

pk The farm node primary key, that is, the attributes that uniquely identify this node.

port
The port to use for REST or RMI

status
The status of this node

tomcatPort
The port to use for RMI if behind a firewall

The *farm node primary key* has the following properties:

projectName
The MobileFirst runtime related to this farm member

serverId
The server identifier of this farm member

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding runtime is not found.`

500

`An internal error occurred.`

Farm topology members (DELETE)

Unregisters a farm node.

Description

This service removes a farm node By default, the service will remove a farm node only if it is marked as being Down If you want to force the deletion, even if the farm member is marked as being Alive, you should set the force argument to true.

Roles

Users in the following roles are authorized to perform this operation:

- **`worklightadmin`**

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/farm/*server-id*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/farm/farm_member_1?forc
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

`server-id`
The server id of the farm member to remove

Query Parameters

Query parameters are optional.

`force`
Whether the service should unregister a farm member, even if it is marked as being Alive. The default is false.

`locale`
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The status of the unregistration of the farm member

JSON Example

```
{
  "ok" : true,
  "productVersion" : "7.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<remove-farm-member-result
  ok="true"
  productVersion="7.0"/>
```

Response Properties

The response has the following properties:

`ok` Whether the operation was successful.

`productVersion`
The exact product version.

Errors

403

`The user is not authorized to call this service.`

404

`The corresponding farm member is not found.`

500

`An internal error occurred.`

GCM Credentials (DELETE)

Deletes Google Cloud Messaging (GCM) credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

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

Method

`DELETE`

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/gcmConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The delete status of GCM credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_PUSH_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteGCMCredentialsStatus
  status="Success"
  type="DELETE_PUSH_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteGCMCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The GCM credentials are deleted successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

GCM Credentials (GET)

Retrieves Google Cloud Messaging (GCM) credentials of the application with the application ID, environment, and version.

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*/gcmConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The GCM Credentials of the application with the application ID, environment, and version.

JSON Example

```
{
  "apiKey" : "AIzaSyDSJrULbNZZzzZZzzxyX7ZTmnoRLkwiU",
  "productVersion" : "7.1.0",
  "senderId" : "9999999999999",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<gcmCredentials
  apiKey="AIzaSyDSJrULbNZZzzZZzzxyX7ZTmnoRLkwiU"
  productVersion="7.1.0"
  senderId="9999999999999"/>
```

Response Properties

The response has the following properties:

apiKey
The key value received from GCM.

productVersion
The exact product version.

senderId
The project ID received from GCM.

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.
```

GCM Credentials (PUT)

Set Google Cloud Messaging (GCM) credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

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

Method

```
PUT
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/gcmConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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

The payload in JSON format has values for apiKey and senderId.

JSON Example

```
{
  "apiKey" : "AIzaSyDSJrrrrrrrrrrZZZZZZZX7ZTmnoRLkwiU",
  "senderId" : "1099999999999",
}
```

Payload Properties

The payload has the following properties:

apiKey
The key value received from GCM.

senderId
The project ID received from GCM.

Response

The status of set GCM credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_GCM_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<setGCMCredentialsStatus
  status="Success"
  type="SET_GCM_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</setGCMCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The GCM credentials are saved successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

Mediator (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/mediat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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 metadata of the mediator.

JSON Example

```
{
  "productVersion" : "7.1.0",
  "type" : "Google",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<mediators
  productVersion="7.1.0"
  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 or not running.
```

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/mediat
```

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.

Produces

application/json, application/xml, text/xml

Response

The list of all supported mediators for sending notifications.

JSON Example

```
{
  "mediators" : [
    {
      "type" : "Google",
    },
    ...
  ],
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<mediators productVersion="7.1.0">
  <mediators>
    <mediator type="Google"/>
    ...
  </mediators>
</mediators>
```

Response Properties

The response has the following properties:

mediators
The array of mediator metadata

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 or not running.
```

500

```
An internal error occurred.
```

MPNS Credentials (DELETE)

Deletes MPNS credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/mpnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The delete status of MPNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_PUSH_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteMPNSCredentials
  status="Success"
  type="DELETE_PUSH_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteMPNSCredentials>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The MPNS credentials are deleted successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.`

MPNS Credentials (GET)

Retrieves MPNS credentials of the application with the application ID, environment, and version.

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*/mpnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The MPNS Credentials of the application with the application ID, environment, and version.

JSON Example

```
{
  "authenticated" : true,
  "keyAlias" : "aliasName",
  "keyAliasPassword" : "password",
  "productVersion" : "7.1.0",
  "serviceName" : "wl.ibm.push",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<mpnsCredentials
  authenticated="true"
  keyAlias="aliasName"
  keyAliasPassword="password"
  productVersion="7.1.0"
  serviceName="wl.ibm.push"/>
```

Response Properties

The response has the following properties:

authenticated
Returns whether the push configuration is authenticated.

keyAlias
The alias used to access the keystore specified in the worklight.properties.

keyAliasPassword
The password for the key alias.

productVersion
The exact product version.

serviceName
The common name (CN) found in the MPNS certificate's Subject value.

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.
```

MPNS Credentials (PUT)

Set MPNS credentials of the application with the application ID, environment, version, keyAlias, keyAliasPassword, and serviceName.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/mpnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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

The payload in JSON format has values for keyAlias, keyAliasPassword, and serviceName.

JSON Example

```
{
  "authenticated" : true,
  "keyAlias" : "aliasName",
  "keyAliasPassword" : "password",
  "serviceName" : "wl.ibm.push",
}
```

Payload Properties

The payload has the following properties:

`authenticated`
Returns whether the push configuration is authenticated.

`keyAlias`
The alias is used to access the keystore that is specified in the worklight.properties file.

`keyAliasPassword`
The password for your key alias.

`serviceName`
The common name (CN) found in the MPNS certificate's Subject value.

Response

The status of set MPNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_MPNS_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<setMPNSCredentialsStatus
  status="Success"
  type="SET_MPNS_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</setMPNSCredentialsStatus>
```

Response Properties

The response has the following properties:

`productVersion`
The exact product version.

project
Project name.

status
The MPNS credentials are saved successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.`

Push Device Registration (DELETE)

Deletes the device with the device ID and application ID.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

`DELETE`

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/devices/*device-id*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

locale
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

Deletes the device with the device ID and application ID.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "REMOVE_DEVICE",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteDevice
  status="Success"
  type="REMOVE_DEVICE">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteDevice>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The device is deleted successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

Push Device Registration (GET)

Retrieves metadata of the device with the given device ID.

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*/devices/*device-id*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

locale
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The metadata of the device with the given device ID.

JSON Example

```
{
  "deviceId" : "testdevice",
  "platform" : "G",
  "productVersion" : "7.1.0",
  "token" : "testtoken",
  "userId" : "worklight",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<devices
  deviceId="testdevice"
  platform="G"
  productVersion="7.1.0"
  token="testtoken"
  userId="worklight"/>
```

Response Properties

The response has the following properties:

deviceId
The unique id of the device.

platform
The device platform.

productVersion
The exact product version.

token
The unique push token of the device.

userId
The userId of the device.

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.
```

Push Device Subscription (DELETE)

Delete subscriptions of a combination of application, tag name, and device ID.

Description

The subscriptions that are deleted are for a combination of application, tag name, and device ID.

Roles

Users in the following roles are authorized to perform this operation:

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

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/subscriptions

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

`application-name`
The name of the application.

Query Parameters

Query parameters are optional.

`deviceId`
The unique deviceId of the device.

`locale`
The locale used for error messages.

`tag-Name`
The tag name.

Produces

application/json, application/xml, text/xml

Response

The status of delete subscriptions.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_SUBSCRIPTIONS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteSubscriptionsStatus
  status="Success"
  type="DELETE_SUBSCRIPTIONS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteSubscriptionsStatus>
```

Response Properties

The response has the following properties:

`productVersion`
The exact product version.

`project`
Project name.

`status`
The subscription is deleted successfully.

`type`
Transaction type.

The *productVersion* has the following properties:

`productVersion`
The exact product version

The *project* has the following properties:

`name`
Name of the project

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.`

Push Device Subscription (GET)

Retrieves metadata of the subscriptions.

Description

The subscriptions can be obtained for application, for a particular tag, for a particular userId, for a particular deviceId and a combination of application, tag name, userId 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

`application-name`
The name of the application.

Query Parameters

Query parameters are optional.

`deviceId`
The unique id of the device.

`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.

size
The number of elements to be returned.

tag-Name
The name of the tag.

userId
The user id.

Produces

application/json, application/xml, text/xml

Response

The metadata of the subscriptions.

JSON Example

```
{
  "offset" : 1,
  "productVersion" : "7.1.0",
  "size" : 6,
  "subscriptions" : [
    {
      "deviceId" : "testdevice",
      "tag-Name" : "testtag",
    },
    ...
  ],
  "totalListSize" : 6,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushDevicesubscription
  offset="1"
  productVersion="7.1.0"
  size="6"
  totalListSize="6">
  <subscriptions>
    <subscription
      deviceId="testdevice"
      tag-Name="testtag"/>
    ...
  </subscriptions>
</pushDevicesubscription>
```

Response Properties

The response has the following properties:

offset
The offset from the beginning of the list if only a part of the list (a page) should be returned.

productVersion
The exact product version.

size
The number of elements to be returned.

subscriptions
The array of subscription metadata

totalListSize
The total number of subscriptions.

The *pushDevicesubscription* has the following properties:

deviceId
The unique id of the device.

tag-Name
The name of the tag.

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.`

Push Devices Registration (GET)

Retrieves metadata for the list of devices of an 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*/notifications/applications/*application-name*/devices

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

offset
The offset from the beginning of the list if only a part of the list (a page) should be returned.

size
The number of elements to be returned.

Produces

application/json, application/xml, text/xml

Response

The metadata of the devices of an application.

JSON Example

```
{
  "devices" : [
    {
      "deviceId" : "testdevice",
      "platform" : "G",
      "token" : "testtoken",
      "userId" : "worklight",
    },
    ...
  ],
  "offset" : 1,
  "productVersion" : "7.1.0",
  "size" : 6,
  "totalListSize" : 6,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<devices
  offset="1"
  productVersion="7.1.0"
  size="6"
  totalListSize="6">
  <devices>
    <device
      deviceId="testdevice"
      platform="G"
      token="testtoken"
      userId="worklight"/>
    ...
  </devices>
</devices>
```

Response Properties

The response has the following properties:

devices
The array of device metadata

offset
The offset from the beginning of the list if only a part of the list (a page) should be returned.

productVersion
The exact product version.

size
The number of elements to be returned.

totalListSize
The total number of deivces.

The *pushDeviceRegistration* has the following properties:

deviceId
The unique id of the device.

platform
The device platform.

token
The unique push token of the device.

userId
The userId of the device.

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.`

Push Enabled Applications (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

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.

Produces

application/json, application/xml, text/xml

Response

The metadata of the deployed push enabled applications.

JSON Example

```
{
  "applications" : [
    {
      "applicationEnvironments" : [
        {
          "deviceCount" : 1,
          "mediatorType" : "Google",
          "numberOfMessagesSent" : 1,
          "userCount" : 1,
        },
        ...
      ],
      "deviceCount" : 1,
      "displayName" : "SampleApplication",
      "name" : "SampleApplication",
      "userCount" : 1,
    },
    ...
  ],
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<pushnotification productVersion="7.1.0">
  <applications>
    <application
      deviceCount="1"
      displayName="SampleApplication"
      name="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 metadata

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 display name of the application.

name
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 or not running.`

500

`An internal error occurred.`

Push Tags (DELETE)

Deletes tag of the application with the application ID and tag.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

`DELETE`

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags/*tag-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

application-name
The name of the application.

tag-name
The name of the tag.

Query Parameters

Query parameters are optional.

locale
The locale used for error messages.

Produces

application/json, application/xml, text/xml

Response

The delete status of delete tag.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_TAGS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteTagStatus
  status="Success"
  type="DELETE_TAGS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteTagStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The tags are deleted successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.`

Push Tags (GET)

Retrieves tags of the application with the application ID.

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*/tags

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

`runtime-name`
The name of the runtime. This is the context 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 Tags of the application with details such as description, name, and product version.

JSON Example

```
{
  "description" : "This is a Gold tag.",
  "name" : "Gold",
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<tags
  description="This is a Gold tag."
  name="Gold"
  productVersion="7.1.0"/>
```

Response Properties

The response has the following properties:

`description`
The description of the Tag.

name
The name of the Tag.

productVersion
The exact product version.

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.`

Push Tags (POST)

Create Tags of the application with the application ID.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

POST

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

The payload in JSON format has values for name and description.

JSON Example

```
{
  "description" : "This is a Gold tag.",
  "name" : "Gold",
}
```

Payload Properties

The payload has the following properties:

description
: The description of the tag.

name
: The name of the tag.

Response

The status of create tags.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "CREATE_TAGS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<createTagsStatus
  status="Success"
  type="CREATE_TAGS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</createTagsStatus>
```

Response Properties

The response has the following properties:

productVersion
: The exact product version.

project
Project name.

status
The tags are created successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

Push Tags (PUT)

Update Tags of the application with the application ID and tag.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
PUT
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/tags/*tag-name*

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context root of the runtime web application, without the leading slash.

application-name
The name of the application.

tag-name
The name of the tag.

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 in JSON format has values for name and description.The tag description is updated based on the given tag name.

JSON Example

```
{
  "description" : "This is modified description of the Gold tag.",
  "name" : "Gold",
}
```

Payload Properties

The payload has the following properties:

description
The description of the tag.

name
The name of the tag.

Response

The status of update tags.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "UPDATE_TAGS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<updateTagsStatus
  status="Success"
  type="UPDATE_TAGS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</updateTagsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The tags are updated successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.`

Runtime (DELETE)

Deletes a specific runtime.

Description

The purpose of this API is to allow to cleanup the database. You can delete a runtime only 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime?locale=de_DE&mode=em
```

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.

mode
Whether to delete the runtime only if it has no applications or adapters. Possible values are `empty` (delete only when empty) and `always` (delete even when not empty, the default).

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. Possible reasons: It is still running, hence you must
It is not empty but you passed the mode empty to delete only an empty runtime.
```

500

```
An internal error occurred.
```

Runtime (GET)

Retrieves metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime?expand=true&locale=de_D
```

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.

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 metadata for the runtime.

JSON Example

```
{
  "adapters" : [
    {
      "deployTime" : "2014-04-13T00:18:36.979Z",
      "description" : "My first sample adapter",
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapte
      "name" : "myadapter",
      "platformVersion" : "6.1.0.00.20131126-0630",
      "procedureDetails" : [
        {
          "audit" : false,
          "connectAs" : "SERVER",
          "description" : "Returns something.",
          "displayName" : "getSomething",
          "name" : "getSomething",
          "securityTest" : "mobileTest",
        },
        ...
      ],
      "projects" : [
        {
          "name" : "myproject",
        },
        ...
      ],
      "resourceSize" : 1024,
      "urls" : [
        {
          "formParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "headerParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "javaClass" : "com.example.MyRestWrapper",
          "javaMethodName" : "multiplyNumbers",
          "method" : "POST",
          "pathParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "queryParameters" : [
            {
              "defaultValue" : "n/a",
              "javaType" : "java.lang.String",
              "name" : "param",
            },
            ...
          ],
          "securedWithScope" : "wl_antiXSRFRealm wl_anonymousUserRealm",
          "uri" : "/multiply",
        },
        ...
      ],
    },
    ...
  ],
  "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.",
          },
          "authenticityConfig" : "BASIC",
          "buildTime" : "2014-03-29T00:18:36.979Z",
          "deployTime" : "2014-04-13T00:18:36.979Z",
          "deviceProvisioningRealm" : "myProvRealm",
          "envPlatformVersion" : "7.1.0",
          "environment" : "android",
          "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/ap
          "prevBuildTime" : "2014-03-29T00:18:36.979Z",
          "resourceSize" : 5120,
          "securityTest" : "mobileTest",
          "supportRemoteDisable" : true,
          "supportsAuthenticity" : true,
          "userAuthenticationRealm" : "myAuthRealm",
          "version" : "1.0",
          "versionLocked" : false,
        },
        ...
      ],
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
      "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" : "7.1.0",
  "running" : true,
  "serverVersion" : "7.1.0",
  "synchronizationStatus" : "ok",
  "topology" : "STANDALONE",
}
```

XML Example

```
<?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="7.1.0"
  running="true"
  serverVersion="7.1.0"
  synchronizationStatus="ok"
  topology="STANDALONE">
  <adapters>
    <adapter
      deployTime="2014-04-13T00:18:36.979Z"
      description="My first sample adapter"
      link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/adapters/my
```

```
        name="myadapter"
        platformVersion="6.1.0.00.20131126-0630"
        resourceSize="1024">
        <procedureDetails>
          <procedureDetail
            audit="false"
            connectAs="SERVER"
            description="Returns something."
            displayName="getSomething"
            name="getSomething"
            securityTest="mobileTest"/>
          ...
        </procedureDetails>
        <projects>
          <project name="myproject"/>
          ...
        </projects>
        <urls>
          <url
            javaClass="com.example.MyRestWrapper"
            javaMethodName="multiplyNumbers"
            method="POST"
            securedWithScope="wl_antiXSRFRealm wl_anonymousUserRealm"
            uri="/multiply">
            <formParameters>
              <formParameter
                defaultValue="n/a"
                javaType="java.lang.String"
                name="param"/>
              ...
            </formParameters>
            <headerParameters>
              <headerParameter
                defaultValue="n/a"
                javaType="java.lang.String"
                name="param"/>
              ...
            </headerParameters>
            <pathParameters>
              <pathParameter
                defaultValue="n/a"
                javaType="java.lang.String"
                name="param"/>
              ...
            </pathParameters>
            <queryParameters>
              <queryParameter
                defaultValue="n/a"
                javaType="java.lang.String"
                name="param"/>
              ...
            </queryParameters>
          </url>
          ...
        </urls>
      </adapter>
      ...
    </adapters>
    <applications>
      <application
        description="My first sample application"
        displayName="My Sample Application"
        link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applicat
        name="myapplication"
        platformVersion="6.1.0.00.20131126-0630">
        <environments>
          <environment
            authenticityConfig="BASIC"
            buildTime="2014-03-29T00:18:36.979Z"
            deployTime="2014-04-13T00:18:36.979Z"
            deviceProvisioningRealm="myProvRealm"
            envPlatformVersion="7.1.0"
            environment="android"
            link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/applica
            prevBuildTime="2014-03-29T00:18:36.979Z"
            resourceSize="5120"
            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 context 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 IBM MobileFirst Platform Foundation 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 IBM MobileFirst Platform Server version number from which `worklight-jee-library.jar` is taken.

synchronizationStatus
The status of the nodes of the runtime. Can contain the values "ok" if all nodes of the runtime are running without error, "synchronizing" if some node is in progress of synchronizing, or an error message if some nodes failed to synchronize.

topology
Server topology. Can contain the values "STANDALONE", "CLUSTER" or "FARM"

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 IBM MobileFirst Platform Foundation development tools (Studio) that built the adapter.

procedureDetails
The JavaScript procedures of the adapter.

projects
The projects the adapter belong to.

resourceSize
The size of the adapter resource.

urls
The API documentation of the URLs of the REST API provided by the adapter.

The *adapter procedure details* has the following properties:

audit
`true` if calls to the Javascript procedure are logged in the audit log.

connectAs
The optional type of the connection to the back end. Possible values are: SERVER (use the connection policy defined for the adapter) or END_USER (use the user's identity for the connection).

description
The optional description of the procedure.

displayName
The optional display name of the procedure.

name
The Javascript name of the adapter procedure.

securityTest
The optional name of the security test used to protect the adapter procedure, as defined in the authenticationConfig.xml file.

The *project* has the following properties:

name
The name of the project, which is the context root of the runtime.

The *url* has the following properties:

formParameters
The form parameters.

headerParameters
The header parameters.

javaClass
The Java class

javaMethodName
The Java method

method
The HTTP method.

pathParameters
The path parameters.

queryParameters
The query parameters.

securedWithScope
The set of realms that secure the REST API.

uri
The URI of the REST API.

The *REST API parameter* has the following properties:

defaultValue
The default value.

javaType
The Java type name.

name
The name of the parameter.

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 IBM MobileFirst Platform Foundation 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.

authenticityConfig
The application authenticity configuration. Possible values are: `NONE`, `BASIC`, `EXTENDED`.

buildTime
The time stamp when the app version was built.

deployTime
The date in ISO 8601 format when the application was deployed.

deviceProvisioningRealm
The name of the realm used for device provisioning.

envPlatformVersion
The version of the platform of the environment.

environment
The platform environment of the app version: android, iphone, ...

link
The URL to access detail information about the application version.

prevBuildTime
The time stamp when the app that was previosuly deployed was built.

publishUrl
For web applications, this is the URL under which the web application was published.

resourceSize
The size of the wlapp file of the application version.

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 authentication.

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/lock?locale=de_DE
```

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.

Produces

application/json, application/xml, text/xml

Response

JSON Example

```
{
  "busy" : false,
}
```

XML Example

```
<?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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/lock?locale=de_DE
```

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.

Produces

application/json, application/xml, text/xml

Response

JSON Example

```
{
  "busy" : true,
}
```

XML Example

```
<?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 metadata 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.example.com/worklightadmin/management-apis/1.0/runtimes?fullInfo=true&locale=de_DE&moc
```

Query Parameters

Query parameters are optional.

fullInfo
The default value `false` returns basic list of properties of a runtime, while the value `true` returns all properties of this runtime (see a GET on a single runtime for the complete list of properties).

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 database that might not be running.

Produces

application/json, application/xml, text/xml

Response

The metadata for the list of runtimes.

JSON Example

```
{
  "productVersion" : "7.1.0",
  "projects" : [
    {
      "auditEnabled" : true,
      "link" : "https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime",
      "name" : "myruntime",
      "numberOfActiveDevices" : 100,
      "numberOfAdapters" : 1,
      "numberOfApplications" : 1,
      "numberOfDecommisionedDevices" : 5,
      "running" : true,
      "synchronizationStatus" : "ok",
      "topology" : "STANDALONE",
    },
    ...
  ],
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<projectconfiguration productVersion="7.1.0">
  <projects>
    <project
      auditEnabled="true"
      link="https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime"
      name="myruntime"
      numberOfActiveDevices="100"
      numberOfAdapters="1"
      numberOfApplications="1"
      numberOfDecommisionedDevices="5"
      running="true"
      synchronizationStatus="ok"
      topology="STANDALONE"/>
    ...
  </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:

`auditEnabled`
Whether audit is enabled.

link
The URL to access detail information about the runtime.

name
The name of the runtime. This is the context 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.

synchronizationStatus
The status of the nodes of the runtime. Can contain the values "ok" if all nodes of the runtime are running without error, "synchronizing" if some node is in progress of synchronizing, or an error message if some nodes failed to synchronize.

topology
Server topology. Can contain the values "STANDALONE", "CLUSTER" or "FARM"

Errors

403

`The user is not authorized to call this service.`

500

`An internal error occurred.`

Send Bulk Messages (POST)

Send bulk messages with different options to be specified.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

`POST`

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages/bulk

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

The payload in JSON format has values for array of messages, target, and settings.

JSON Example

```
{
  "//ArrayOfMessageBody" : [
    {
      "messages" : {
        "alert" : "Test message",
      },
      "settings" : {
        "apns" : {
          "badge" : 1,
          "iosActionKey" : "Ok",
          "payload" : "",
          "sound" : "song.mp3",
        },
        "gcm" : {
          "delayWhileIdle" : ,
          "payload" : "",
          "sound" : "song.mp3",
          "timeToLive" : ,
        },
        "mpns" : {
          "raw" : {
            "payload" : {
            },
          },
          "title" : {
            "backBackgroundImage" : "Blue.jpg",
            "backContent" : "Back Title Content",
            "backTitle" : "Back Title",
            "backgroundImage" : "Red.jpg",
            "count" : 1,
            "title" : "Push Notification",
          },
          "toast" : {
            "param" : "/Page2.xaml?NavigatedFrom=Toast Notification",
            "title" : "Hello",
          },
        },
      },
      "target" : {
        "consumerIds" : [ "MyConsumerId1", ... ],
        "deviceIds" : [ "MyDeviceId1", ... ],
        "platforms" : [ "A,G", ... ],
        "tagNames" : [ "Gold", ... ],
      },
    },
    ...
  ],
}
```

Payload Properties

The payload has the following properties:

//ArrayOfMessageBody
The array of message

The *bulk-messages* has the following properties:

messages
The array of message

settings
The settings are the different attributes of the notification.

target
Set of targets can be consumer Ids, devices, platforms, or tags.

The *message* has the following properties:

alert
A string to be displayed in the alert.

The *settings* has the following properties:

apns
Attributes for sending message to an iOS device.

gcm
Attributes for sending message to an Android device.

mpns
Attributes for sending message to an MPNS device.

The *apns* has the following properties:

badge
An integer value to be displayed in a badge on the application icon.

iosActionKey
The label of the dialog box button that allows the user to open the app upon receiving the notification.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a file to play when the notification arrives.

The *gcm* has the following properties:

delayWhileIdle
A Boolean value that indicates that the message must not be sent if the device is idle. The server waits for the device to become active before the message is sent.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a file to play when the notification arrives.

timeToLive
The duration (in seconds) that the message is kept on GCM storage if the device is offline. Default value is 4 weeks, and must be set as a JSON number.

The *mpns* has the following properties:

raw
Raw.

title
Title.

toast
Toast.

The *raw* has the following properties:

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

The *title* has the following properties:

backBackgroundImage
URL of the back image on a medium flip Tile.

backContent
Content that displays on the back of a medium flip Tile.

backTitle
Title that displays on the back of a flip Tile.

backgroundImage
URL of the front image on a medium flip Tile.

count
An integer value from 1 to 99. If the value of count is not set or it is set to 0, the circle image and value do not display in the Tile. This property is also known as badge.

title
A string that indicates the title of the application. The title fits on a single line of text and must not be wider than the Tile. Approximately 15 characters can fit in the title before it is truncated.

The *toast* has the following properties:

param
Toast notification content.

title
Toast notification title.

The *target* has the following properties:

consumerIds
The array of consumer Ids.

deviceIds
JSON array of the device ids. Devices with these ids receive the notification.

platforms
JSON array of platforms. Devices running on these platforms receive the notification. Supported values are A, G, and M.

tagNames
JSON array of tags. Devices that are subscribed to these tags receive the notification.

Response

The status of send messages.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_PUSH_NOTIFICATION_ENABLED",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<sendMessagesStatus
  status="Success"
  type="SET_PUSH_NOTIFICATION_ENABLED">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</sendMessagesStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
Message submitted for delivery.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

Send Message (POST)

Send message with different options to be specified.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
POST
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/messages

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

The payload in JSON format has values for message, target, and settings.

JSON Example

```
{
  "message" : {
    "alert" : "Test message",
  },
  "settings" : {
    "apns" : {
      "badge" : 1,
      "iosActionKey" : "Ok",
      "payload" : "",
      "sound" : "song.mp3",
    },
    "gcm" : {
      "delayWhileIdle" : ,
      "payload" : "",
      "sound" : "song.mp3",
      "timeToLive" : ,
    },
    "mpns" : {
      "raw" : {
        "payload" : {
        },
      },
      "title" : {
        "backBackgroundImage" : "Blue.jpg",
        "backContent" : "Back Title Content",
        "backTitle" : "Back Title",
        "backgroundImage" : "Red.jpg",
        "count" : 1,
        "title" : "Push Notification",
      },
      "toast" : {
        "param" : "/Page2.xaml?NavigatedFrom=Toast Notification",
        "title" : "Hello",
      },
    },
  },
  "target" : {
    "consumerIds" : [ "MyConsumerId1", ... ],
    "deviceIds" : [ "MyDeviceId1", ... ],
    "platforms" : [ "A,G", ... ],
    "tagNames" : [ "Gold", ... ],
  },
}
```

Payload Properties

The payload has the following properties:

message
The alert message to be sent

settings
The settings are the different attributes of the notification.

target
Set of targets can be consumer Ids, devices, platforms, or tags.

The *message* has the following properties:

alert
A string to be displayed in the alert.

The *settings* has the following properties:

apns
Attributes for sending message to an iOS device.

gcm
Attributes for sending message to an Android device.

mpns
Attributes for sending message to an MPNS device.

The *apns* has the following properties:

badge
An integer value to be displayed in a badge on the application icon.

iosActionKey
The label of the dialog box button that allows the user to open the app upon receiving the notification.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a file to play when the notification arrives.

The *gcm* has the following properties:

delayWhileIdle
A Boolean value that indicates that the message must not be sent if the device is idle. The server waits for the device to become active before the message is sent.

payload
: A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
: The name of a file to play when the notification arrives.

timeToLive
: The duration (in seconds) that the message is kept on GCM storage if the device is offline. Default value is 4 weeks, and must be set as a JSON number.

The *mpns* has the following properties:

raw
: Raw.

title
: Title.

toast
: Toast.

The *raw* has the following properties:

payload
: A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

The *title* has the following properties:

backBackgroundImage
: URL of the back image on a medium flip Tile.

backContent
: Content that displays on the back of a medium flip Tile.

backTitle
: Title that displays on the back of a flip Tile.

backgroundImage
: URL of the front image on a medium flip Tile.

count
: An integer value from 1 to 99. If the value of count is not set or it is set to 0, the circle image and value do not display in the Tile. This property is also known as badge.

title
: A string that indicates the title of the application. The title fits on a single line of text and must not be wider than the Tile. Approximately 15 characters can fit in the title before it is truncated.

The *toast* has the following properties:

param
: Toast notification content.

title
: Toast notification title.

The *target* has the following properties:

consumerIds
The array of consumer Ids.

deviceIds
JSON array of the device ids. Devices with these ids receive the notification.

platforms
JSON array of platforms. Devices running on these platforms receive the notification. Supported values are A, G, and M.

tagNames
JSON array of tags. Devices that are subscribed to these tags receive the notification.

Response

The status of send message.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_PUSH_NOTIFICATION_ENABLED",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<sendMessageStatus
  status="Success"
  type="SET_PUSH_NOTIFICATION_ENABLED">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</sendMessageStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
Message submitted for delivery.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/transactions/1?local
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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.

JSON Example

```
{
  "appServerId" : "Tomcat",
  "description" : {
  },
  "errors" : [
    {
      "details" : "An internal error occured.",
    },
    ...
  ],
  "id" : 1,
  "productVersion" : "7.1.0",
  "project" : {
    "name" : "myproject",
  },
  "status" : "FAILURE",
  "timeCreated" : "2014-04-13T00:18:36.979Z",
  "timeUpdated" : "2014-04-14T00:18:36.979Z",
  "type" : "DELETE_ADAPTER",
  "userName" : "demouser",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<transaction
  appServerId="Tomcat"
  id="1"
  productVersion="7.1.0"
  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 failed transactions.

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/errors

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/transactions/errors?bo
```

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.

bookmark
The bookmark for the page if only a part of the list (a page) should be returned. If a bookmark is specified, the offset parameter is 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.

JSON Example

```
{
  "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",
    },
    ...
  ],
  "nextPageBookmark" : "DEF",
  "pageNumber" : 2,
  "pageSize" : 100,
  "prevPageBookmark" : "ABC",
  "productVersion" : "7.1.0",
  "startIndex" : 0,
  "totalListSize" : 33,
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<transactions
  nextPageBookmark="DEF"
  pageNumber="2"
  pageSize="100"
  prevPageBookmark="ABC"
  productVersion="7.1.0"
  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

nextPageBookmark
The bookmark of the next page if only a page of transactions is returned.

pageNumber
The page index if only a page of transactions is returned.

pageSize
The page size if only a page of transactions is returned.

prevPageBookmark
The bookmark of the previous page 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.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/unsubs
```

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.

Consumes

application/json

Produces

application/json, application/xml, text/xml

Payload

The payload with comma separated list of phone numbers.

JSON Example

```
{
  "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.

JSON Example

```
{
  "failure" : "5678",
  "success" : "1234",
}
```

XML Example

```
<?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 or not running.
```

500

```
An internal error occurred.
```

WNS Credentials (DELETE)

Deletes Windows Push Notification Services (WNS) credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

```
DELETE
```

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/ *application-name*/wnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The delete status of WNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "DELETE_PUSH_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<deleteWNSCredentialsStatus
  status="Success"
  type="DELETE_PUSH_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</deleteWNSCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The WNS credentials are deleted successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

WNS Credentials (GET)

Retrieves Windows Push Notification Services (WNS) credentials of the application with the application ID, environment, and version.

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*/wnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applic
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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/json, application/xml, text/xml

Response

The WNS Credentials of the application with the application ID, environment, and version.

JSON Example

```
{
  "clientSecret" : "82e4569er",
  "packageSID" : "ms-app://s-123-566-78910",
  "productVersion" : "7.1.0",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<wnsCredentials
  clientSecret="82e4569er"
  packageSID="ms-app://s-123-566-78910"
  productVersion="7.1.0"/>
```

Response Properties

The response has the following properties:

clientSecret
The client secret received from WNS.

packageSID
The unique identifier of the app received from WNS.

productVersion
The exact product version.

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.`

WNS Credentials (PUT)

Sets Windows Push Notification Services (WNS) credentials of the application with the application ID, environment, and version.

Roles

Users in the following roles are authorized to perform this operation:

- **worklightadmin**
- **worklightdeployer**
- **worklightmonitor**
- **worklightoperator**

Method

PUT

Path

/management-apis/1.0/runtimes/*runtime-name*/notifications/applications/*application-name*/wnsConf/*application-env*/*application-version*/

Example

```
https://www.example.com/worklightadmin/management-apis/1.0/runtimes/myruntime/notifications/applicat
```

Path Parameters

runtime-name
The name of the runtime. This is the context 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

The payload in JSON format has values for packageSID and clientSecret.

JSON Example

```
{
  "clientSecret" : "82e4569er",
  "packageSID" : "ms-app://s-123-566-78910",
}
```

Payload Properties

The payload has the following properties:

clientSecret
The client secret received from WNS.

packageSID
The unique identifier of the app received from WNS.

Response

The status of set WNS credentials.

JSON Example

```
{
  "productVersion" : {
    "productVersion" : "7.1.0",
  },
  "project" : {
    "name" : "PushNotifications",
  },
  "status" : "Success",
  "type" : "SET_WNS_CREDENTIALS",
}
```

XML Example

```
<?xml version="1.0" encoding="UTF-8"?>
<setWNSCredentialsStatus
  status="Success"
  type="SET_WNS_CREDENTIALS">
  <productVersion productVersion="7.1.0"/>
  <project name="PushNotifications"/>
</setWNSCredentialsStatus>
```

Response Properties

The response has the following properties:

productVersion
The exact product version.

project
Project name.

status
The WNS credentials are saved successfully.

type
Transaction type.

The *productVersion* has the following properties:

productVersion
The exact product version

The *project* has the following properties:

name
Name of the project

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.
```

REST API Runtime Services

The REST API for Push in the MobileFirst runtime environment enables back-end server applications that were deployed outside of the MobileFirst Server to access Push functions from a REST API endpoint.

The Push service on the MobileFirst Server is exposed over a REST API endpoint that can be directly accessed by non-mobile clients. You can use the REST API runtime services for Push for registrations, subscriptions, messages, and retrieving tags. Paging and filtering is supported for database persistence in both Cloudant and SQL.

This REST API endpoint is protected by OAuth which requires the clients to be confidential clients and also possess the required access scopes in their OAuth access tokens that is passed by a designated HTTP header.

Push Device Registration (DELETE)

Deletes(Unregisters) an existing device registration of Push

Description

The device registrations of push service is deleted for the given deviceId. The call returns HTTP response code 204 with no content on successful deletion of the device rgistration.

Method

DELETE

Path

/apps/*applicationId*/devices/*deviceId*

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/devices/12345-6789
```

Path Parameters

deviceId
The device identifier

applicationId
The name or identifier of the application

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages.. Default:en-US

Authorization
The token with the scope "unregister-device" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request..

mfpEnvironmentVersion
(Optional) The application version to use while executing the request..

Produces

application/json

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
A device registration with the specified deviceId is not found.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push Device Registration (GET)

Retrieves all or a subset of existing device registration(s) of Push.

Description

The device registrations of push service is retrieved for the given criteria.

Method

```
GET
```

Path

/apps/*applicationId*/devices

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/devices?expand=true&filter=platform==A&offse
```

Path Parameters

applicationId
The name or identifier of the application

Query Parameters

Query parameters are optional.

expand
Retrieves additional metadata for every device registration that is returned in the response.

filter
The filter specifies the search criteria. Refer to the filter section for detailed syntax.

offset
Pagination offset that is normally used along with the size.

size
Pagination size that is normally used along with the offset to retrieve a subset.

userId
Retrieves subscriptions only for the specified userId.

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages.. Default:en-US

Authorization
The token with the scope "get-devices" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request..

mfpEnvironmentVersion
(Optional) The application version to use while executing the request..

Produces

application/json

Response

The details of the device registration that is retrieved.

JSON Example

```
{
  "devices" : [
    {
      "deviceId" : "12345-6789",
      "platform" : "A",
      "token" : "12345-6789",
      "userId" : "admin",
    },
    ...
  ],
  "pageInfo" : {
    "count" : "2",
    "next" : "",
    "previous" : "",
    "totalCount" : "10",
  },
}
```

Response Properties

The response has the following properties:

devices
The array of device registrations with Push.

pageInfo
The pagination information

The *devices* has the following properties:

deviceId
The unique id of the device.

platform
The device platform.

token
The unique push token of the device.

userId
The userId of the device.

The *pageInfo* has the following properties:

count
The number of device registration that are retrieved

next
A hyperlink to the next page

previous
A hyperlink to the previous page

totalCount
The total number of device registration present for the given search criteria

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Device Subscription (DELETE)

Unsubscribes the speficied device from a tag.

Description

Given the deviceId and the tag name, the request deletes a existing subscription from a tag specified. It will return HTTP response code 204 with no content on successfully unsubscribing

Method

```
DELETE
```

Path

/apps/*applicationId*/subscriptions

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/subscriptions?deviceId=12345-6789&tagName=tes
```

Path Parameters

`applicationId`
The name or identifier of the application

Query Parameters

Query parameters are optional.

`deviceId`
The unique identifier for the device

`tagName`
The tag name to unsubscribe

Header Parameters

Some header parameters are optional.

`Accept-Language`
(Optional) The preferred language to use for error messages. Default:en-US

`Authorization`
The token with the scope "unsubscribe-tag" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request.

mfpEnvironmentVersion
(Optional) The application version to use while executing the request.

Produces

application/json

Errors

400

```
A device registraion has userId longer than 254 characters.
```

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
A device registraion with the specified deviceId is not found.
```

405

```
Unsupported Content type - The content type specified in Content-Type header is not application/js
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push Device Subscription (GET)

Retrieves all or a subset of existing subscriptions

Description

The subscriptions of push service is retrieved for the given criteria.

Method

```
GET
```

Path

/apps/*applicationId*/subscriptions

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/subscriptions?deviceId=12345-6789&expand=t
```

Path Parameters

applicationId
The name or identifier of the application

Query Parameters

Query parameters are optional.

deviceId
Retrives subscriptions only for the specified deviceId

expand
Retrives additional metadata for every subscription that is returned in the response

filter
The filter specifies the search criteria. Refer to the filter section for detailed syntax

offset
Pagination offset that is normally used along with the size

size
Pagination size that is normally used along with the offset to retrieve a subset

tagName
Retrives subscriptions only for the specified tagName

userId
Retrives subscriptions only for the specified userId

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "get-subscriptions" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request.

mfpEnvironmentVersion
(Optional) The application version to use while executing the request.

Produces

application/json

Response

The details of the device registration that is retrieved.

JSON Example

```
{
  "devices" : [
    {
      "deviceId" : "12345-6789",
      "platform" : "A",
      "token" : "12345-6789",
      "userId" : "admin",
    },
    ...
  ],
  "pageInfo" : {
    "count" : "2",
    "next" : "",
    "previous" : "",
    "totalCount" : "10",
  },
}
```

Response Properties

The response has the following properties:

devices
The array of device registrations with Push.

pageInfo
The pagination information

The *devices* has the following properties:

deviceId
The unique id of the device.

platform
The device platform.

token
The unique push token of the device.

userId
The userId of the device.

The *pageInfo* has the following properties:

count
The number of device registration that are retrieved

next
A hyperlink to the next page

previous
A hyperlink to the previous page

totalCount
The total number of device registration present for the given search criteria

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Push Device Subscription (POST)

Creates a new subscription for a tag.

Description

Given the deviceId and the tag name, the request creates a new subscription which subscribes the device to the tag specified

Method

```
POST
```

Path

/apps/*applicationId*/subscriptions

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/subscriptions
```

Path Parameters

applicationId
The name or identifier of the application

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "subscribe-tag" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
Specify the JSON content type. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request.

mfpEnvironmentVersion
(Optional) The application version to use while executing the request.

Consumes

application/json

Produces

application/json

Payload

The details of the device and the tag name to which it has to subscribe.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "tagName" : "testTag",
}
```

Payload Properties

The payload has the following properties:

deviceId
The unique id of the device.

tagName
The tag name to subscribe.

Response

The details of the device registration that is updated.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "tagName" : "testTag",
}
```

Response Properties

The response has the following properties:

deviceId
The unique id of the device.

tagName
The tag name to subscribe.

Errors

400

`A device registraion has userId longer than 254 characters.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A device registraion with the specified deviceId is not found.`

405

`Unsupported Content type - The content type specified in Content-Type header is not application/js`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Push Tags (GET)

Retrieves an existing tag of Push

Method

`GET`

Path

/apps/*applicationId*/tags/*tagName*

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/tags/sports
```

Path Parameters

applicationId
The name or identifier of the application

tagName
The name of the tag

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "get-tags" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Produces

application/json

Response

The details of the tag that is retrieved.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-05-20T11:42:11Z",
  "lastUpdatedTime" : "2015-05-20T11:42:11Z",
  "name" : "sports",
}
```

Response Properties

The response has the following properties:

createdMode
The mode of creation.

createdTime
The date and time when the push device registration was created on the server in ISO 8601 format.

lastUpdatedTime
The date and time when the push device registration was last updated on the server in ISO 8601 format.

name
The name of the tag.

Errors

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A device registraion with the specified deviceId is not found.`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Retrieve a Device Registration (GET)

Retrieves an existing device registration of Push

Description

The device registrations of push service is retrieved for the given deviceId.

Method

```
GET
```

Path

/apps/*applicationId*/devices/*deviceId*

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/devices/12345-6789
```

Path Parameters

`deviceId`
The device identifier

`applicationId`
The name or identifier of the application

Header Parameters

Some header parameters are optional.

`Accept-Language`
(Optional) The preferred language to use for error messages. Default:en-US

`Authorization`
The token with the scope "get-devices" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

`mfpEnvironmentName`
(Optional) The environment name to use while executing the request.

`mfpEnvironmentVersion`
(Optional) The application version to use while executing the request.

Produces

application/json

Response

The details of the device registration that is retrieved.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-05-20T11:42:11Z",
  "deviceId" : "12345-6789",
  "lastUpdatedTime" : "2015-05-20T11:42:11Z",
  "platform" : "A",
  "token" : "12345-6789",
  "userId" : "admin",
}
```

Response Properties

The response has the following properties:

createdMode
The mode of creation.

createdTime
The date and time when the push device registration was created on the server in ISO 8601 format.

deviceId
The unique id of the device.

lastUpdatedTime
The date and time when the push device registration was last updated on the server in ISO 8601 format.

platform
The device platform.

token
The unique push token of the device.

userId
The userId of the device.

Errors

401

```
Unauthorized - The caller is either not authenticated or not authorized to make this request.
```

404

```
A device registration with the specified deviceId is not found.
```

406

```
Unsupported Accept type - The content type specified in Accept header is not application/json.
```

500

```
An internal error occurred.
```

Send Bulk Messages (POST)

Send bulk messages with different options to be specified.

Method

```
POST
```

Path

/apps/*applicationId*/messages/bulk

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/messages/bulk
```

Path Parameters

applicationId
The name or identifier of the application

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "send-bulk-messages" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Consumes

application/json

Produces

application/json

Payload

The payload in JSON format has values for array of messages, target, and settings.

JSON Example

```
{
  "//ArrayOfMessageBody" : [
    {
      "messages" : {
        "alert" : "Test message",
      },
      "settings" : {
        "apns" : {
          "badge" : 1,
          "iosActionKey" : "Ok",
          "payload" : "",
          "sound" : "song.mp3",
          "type" : "SILENT",
        },
        "gcm" : {
          "delayWhileIdle" : ,
          "payload" : "",
          "sound" : "song.mp3",
          "timeToLive" : ,
        },
        "mpns" : {
          "raw" : {
            "payload" : {
            },
          },
          "title" : {
            "backBackgroundImage" : "Blue.jpg",
            "backContent" : "Back Title Content",
            "backTitle" : "Back Title",
            "backgroundImage" : "Red.jpg",
            "count" : 1,
            "title" : "Push Notification",
          },
          "toast" : {
            "param" : "/Page2.xaml?NavigatedFrom=Toast Notification",
            "title" : "Hello",
          },
        },
      },
      "target" : {
        "deviceIds" : [ "MyDeviceId1", ... ],
        "platforms" : [ "A,G", ... ],
        "tagNames" : [ "Gold", ... ],
        "userIds" : [ "MyUserId", ... ],
      },
    },
    ...
  ],
}
```

Payload Properties

The payload has the following properties:

`//ArrayOfMessageBody`
The array of message

The *bulk-messages* has the following properties:

`messages`
The array of message

`settings`
The settings are the different attributes of the notification.

`target`
Set of targets can be userIds, devices, platforms, or tags.

The *message* has the following properties:

`alert`
A string to be displayed in the alert.

The *settings* has the following properties:

`apns`
Attributes for sending message to an iOS device.

`gcm`
Attributes for sending message to an Android device.

mpns
Attributes for sending message to an MPNS device.

The *apns* has the following properties:

badge
An integer value to be displayed in a badge on the application icon.

iosActionKey
The label of the dialog box button that allows the user to open the app upon receiving the notification.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a file to play when the notification arrives.

type
Specify the type of APNS notification. It should be either DEFAULT, MIXED or SILENT

The *gcm* has the following properties:

delayWhileIdle
A Boolean value that indicates that the message must not be sent if the device is idle. The server waits for the device to become active before the message is sent.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a sound file on the device to play when the notification arrives to the device.

timeToLive
The duration (in seconds) that the message is kept on GCM storage if the device is offline. Default value is 4 weeks, and must be set as a JSON number.

The *mpns* has the following properties:

raw
Raw.

title
Title.

toast
Toast.

The *raw* has the following properties:

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

The *title* has the following properties:

backBackgroundImage
URL of the back image on a medium flip Tile.

backContent
Content that displays on the back of a medium flip Tile.

backTitle
Title that displays on the back of a flip Tile.

backgroundImage
URL of the front image on a medium flip Tile.

count
An integer value from 1 to 99. If the value of count is not set or it is set to 0, the circle image and value do not display in the Tile. This property is also known as badge.

title
A string that indicates the title of the application. The title fits on a single line of text and must not be wider than the Tile. Approximately 15 characters can fit in the title before it is truncated.

The *toast* has the following properties:

param
Toast notification content.

title
Toast notification title.

The *target* has the following properties:

deviceIds
An array of the devices represented by the device identifiers. Devices with these ids receive the notification. This is a unicast notification

platforms
An array of device platforms. Devices running on these platforms receive the notification. Supported values are A (Apple/iOS), G (Google/Android) and M (Microsoft/Windows).

tagNames
An array of tags specified as tagNames. Devices that are subscribed to these tags receive the notification. Use this type of target for tag based notifications

userIds
An array of users represented by their userIds to send the notification. This is a unicast notification.

Errors

400

```
Invalid JSON.
```

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.
```

Send Message (POST)

Send message with different options.

Description

Sends a push notifications to the specified targets and returns HTTP return code 202 when the request to send the message is accepted.

Method

```
POST
```

Path

/apps/*applicationId*/messages

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/messages
```

Path Parameters

applicationId
The name or identifier of the application

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "send-message" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Consumes

application/json

Produces

application/json

Payload

The payload in JSON format has values for message, target, and settings.

JSON Example

```
{
  "message" : {
    "alert" : "Test message",
  },
  "settings" : {
    "apns" : {
      "badge" : 1,
      "iosActionKey" : "Ok",
      "payload" : "",
      "sound" : "song.mp3",
      "type" : "SILENT",
    },
    "gcm" : {
      "delayWhileIdle" : ,
      "payload" : "",
      "sound" : "song.mp3",
      "timeToLive" : ,
    },
    "mpns" : {
      "raw" : {
        "payload" : {
        },
      },
      "title" : {
        "backBackgroundImage" : "Blue.jpg",
        "backContent" : "Back Title Content",
        "backTitle" : "Back Title",
        "backgroundImage" : "Red.jpg",
        "count" : 1,
        "title" : "Push Notification",
      },
      "toast" : {
        "param" : "/Page2.xaml?NavigatedFrom=Toast Notification",
        "title" : "Hello",
      },
    },
  },
  "target" : {
    "deviceIds" : [ "MyDeviceId1", ... ],
    "platforms" : [ "A,G", ... ],
    "tagNames" : [ "Gold", ... ],
    "userIds" : [ "MyUserId", ... ],
  },
}
```

Payload Properties

The payload has the following properties:

`message`
: The alert message to be sent

`settings`
: The settings are the different attributes of the notification.

`target`
: Set of targets can be user Ids, devices, platforms, or tags. Only one of the targets can be set.

The *message* has the following properties:

`alert`
: A string to be displayed in the alert.

The *settings* has the following properties:

`apns`
: Attributes for sending message to an iOS device.

`gcm`
: Attributes for sending message to an Android device.

mpns
Attributes for sending message to an MPNS (Windows) device.

The *apns* has the following properties:

badge
An integer value to be displayed in a badge on the application icon.

iosActionKey
The label of the dialog box button that allows the user to open the app upon receiving the notification.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a file to play when the notification arrives.

type
Specify the type of APNS notification. It should be either DEFAULT, MIXED or SILENT

The *gcm* has the following properties:

delayWhileIdle
A Boolean value that indicates that the message must not be sent if the device is idle. The server waits for the device to become active before the message is sent.

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

sound
The name of a sound file on the device to play when the notification arrives to the device.

timeToLive
The duration (in seconds) that the message is kept on GCM storage if the device is offline. Default value is 4 weeks, and must be set as a JSON number.

The *mpns* has the following properties:

raw
Raw.

title
Title.

toast
Toast.

The *raw* has the following properties:

payload
A JSON block that is transferred to the application if the application is opened by the user when the notification is received, or if the application is already open.

The *title* has the following properties:

`backBackgroundImage`
URL of the back image on a medium flip Tile.

`backContent`
Content that displays on the back of a medium flip Tile.

`backTitle`
Title that displays on the back of a flip Tile.

`backgroundImage`
URL of the front image on a medium flip Tile.

`count`
An integer value from 1 to 99. If the value of count is not set or it is set to 0, the circle image and value do not display in the Tile. This property is also known as badge.

`title`
A string that indicates the title of the application. The title fits on a single line of text and must not be wider than the Tile. Approximately 15 characters can fit in the title before it is truncated.

The *toast* has the following properties:

`param`
Toast notification content.

`title`
Toast notification title.

The *target* has the following properties:

`deviceIds`
An array of the devices represented by the device identifiers. Devices with these ids receive the notification. This is a unicast notification

`platforms`
An array of device platforms. Devices running on these platforms receive the notification. Supported values are A (Apple/iOS), G (Google/Android) and M (Microsoft/Windows).

`tagNames`
An array of tags specified as tagNames. Devices that are subscribed to these tags receive the notification. Use this type of target for tag based notifications

`userIds`
An array of users represented by their userIds to send the notification. This is a unicast notification.

Errors

400

`Invalid JSON.`

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.`

Update Device Registrations (PUT)

Updates an existing device registration of Push.

Description

The push device registration is updated with the new user identifier or the token specified. In most use cases this call is used to update the userId only.

Method

```
PUT
```

Path

/apps/*applicationId*/devices/*deviceId*

Example

```
https://example.com:443/myproject/imfpush/v1/apps/myapp/devices/12345-6789
```

Path Parameters

deviceId
The device identifier

applicationId
The name or identifier of the application

Header Parameters

Some header parameters are optional.

Accept-Language
(Optional) The preferred language to use for error messages. Default:en-US

Authorization
The token with the scope "update-device" obtained using the confidential client in the format Bearer *token*.. This parameter has to be mandatorily set.

Content-Type
Specify the JSON content type. For example: application/json. This parameter has to be mandatorily set.

mfpEnvironmentName
(Optional) The environment name to use while executing the request.

mfpEnvironmentVersion
(Optional) The application version to use while executing the request.

Consumes

application/json

Produces

application/json

Payload

The details of the device registration will be updated.

JSON Example

```
{
  "deviceId" : "12345-6789",
  "token" : "xyz",
  "userId" : "admin",
}
```

Payload Properties

The payload has the following properties:

deviceId
The unique id of the device.

token
The token of the device. Its optional to set this.

userId
The userId of the device. Its optional to set this.

Response

The details of the device registration that is updated.

JSON Example

```
{
  "createdMode" : "API",
  "createdTime" : "2015-05-20T11:42:11Z",
  "deviceId" : "12345-6789",
  "lastUpdatedTime" : "2015-05-20T11:42:11Z",
  "platform" : "A",
  "token" : "12345-6789",
  "userId" : "admin",
}
```

Response Properties

The response has the following properties:

createdMode
The mode of creation.

createdTime
The date and time when the push device registration was created on the server in ISO 8601 format.

deviceId
The unique id of the device.

lastUpdatedTime
The date and time when the push device registration was last updated on the server in ISO 8601 format.

platform
The device platform.

token
The unique push token of the device.

`userId`
The userId of the device.

Errors

400

`A device registration has userId longer than 254 characters.`

401

`Unauthorized - The caller is either not authenticated or not authorized to make this request.`

404

`A device registration with the specified deviceId is not found.`

405

`Unsupported Content type - The content type specified in Content-Type header is not application/js`

406

`Unsupported Accept type - The content type specified in Accept header is not application/json.`

500

`An internal error occurred.`

Filter syntax

The filter has the following syntax: `name operator expression`

name The field name on which the filter is being applied.

operator
Either `==` (Exact Match) or `=@` (Contains Substring) that describes filter match to use.

expression
The values to include in the result.

For Cloudant data persistence, do not use the @ character in a filter expression when the operator is *Contains Substring*.

When a comma and a backslash are displayed in an expression, they must be backslash-escaped.

When using multiple filters, they can be combined using `AND` and `OR` logic.

For `AND` logic, use multiple filters in the query.

For `OR` logic, use a comma(,) inside of the filter expression.

For both `AND` and `OR` logic, a single query can have both however each filter is evaluated individually before combining them in an `AND` expression.

For the device GET API the following combinations are supported:

- The name can be one of these fields: **userId** or **platform**
- Except for the platform, the operator can be `==` or `=@`
- For the platform, the operator must be `==`
- If operator `=@` is used, the value can be a sub string. If `==` is used, the value must be an exact matching string.

For the subscription GET API the following combinations are supported:

- The name can be one of these fields: **tagName** or **deviceId 2**
- Except for the platform, the operator can be `==` or `=@`
- For the platform, the operator must be `==`
- If operator `=@` is used, the value can be a sub string. If `==` is used, the value must be an exact matching string.

MobileFirst Cloudant API reference

Cloudant is an advanced NoSQL database that is capable of handling a wide variety of data types, such as JSON, full-text, and geospatial data. You can use the Cloudant APIs to access a Cloudant database from your mobile application.

For more information about the SDK and Cloudant, see “Storing mobile data in Cloudant” on page 8-151.

Objective-C API for MobileFirst Cloudant extensions

Use the Objective-C API reference if you are developing a native iOS application in Swift or Objective-C that accesses a Cloudant database.

You can use the IMFData Framework for iOS and the CloudantToolkit Framework for iOS to make your data persistent in the cloud.

- You can find description of the API of the IMFData Framework for iOS in the following file: Objective-C API for MobileFirst Cloudant extensions: IMFData
- You can find description of the API of the CloudantToolkit Framework for iOS in the following file: Objective-C API for MobileFirst Cloudant extensions: Cloudant Toolkit

Deploying MobileFirst projects

Note: Before you can deploy the project to the production environment, you must install the MobileFirst Administration Components as described in "Installing the MobileFirst Server administration" on page 6-47.

You can deploy several MobileFirst runtime environments (that is, several project WAR files) to an application server just as you would deploy any Java EE application. Each deployed project must have a unique name and a unique context path.

Note: An Administration Service must be installed on the application server for one of the topologies that are listed at "Planning deployment of administration components and runtimes" on page 6-8. Otherwise, the 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 relational 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. For Cloudant, the project name is included in the database name so every project uses a different database.

Several instances of MobileFirst Server with different versions of IBM MobileFirst Platform Foundation for iOS installed can share the same application server and the same MobileFirst Administration Service. However, they must be migrated to be compatible with the current version of the Administration Service. For more information about migration, see "Migrating a project WAR file for use with a new MobileFirst Server" on page 11-37.

Deploying MobileFirst 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

After you have developed an application, you want to move from your development environment and deploy a MobileFirst project to a test or production environment.

Before you begin

You have built a MobileFirst project that contains one or more applications in MobileFirst tools. A WAR file and a set of `.wlapp` files are created in the `bin` folder of your MobileFirst project. You now want to deploy the project and the applications to a test or production environment.

- A WAR file is created by MobileFirst tools for every MobileFirst 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-iphone-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 MobileFirst Server administration components as described in "Installing the MobileFirst Server administration" on page 6-47.

   You can have several MobileFirst runtime environments that are managed by the same MobileFirst Operations Console. Verify that you have deployment rights for IBM MobileFirst Platform Foundation for iOS, such as the role of **`worklightdeployer`** or **`worklightadmin`**. For more information, see "Configuring user authentication for MobileFirst Server administration" on page 6-96.
2. For each application in the project, change the settings in the `application-descriptor.xml` file to match your production environment.If necessary, depending on the functions and licensing provisions of the app, change the following settings.
   - Settings screen
   - Device provisioning
   - Application license type. (Only applicable if IBM MobileFirst Platform Foundation for iOS version 7.1, Fix Pack 1 or later is installed, and token licensing is activated on the MobileFirst Server.) For more information, see "Token licensing: setting the license app type" on page 8-21.
   - Application authenticity
   - User authentication

   For more information about each setting, see "Application descriptor of native API applications" on page 8-7.
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 configuration properties on the server. When you deploy your MobileFirst 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 MobileFirst project in production by using JNDI environment entries" on page 11-65.
4. Create the project WAR file in either of two ways:

- Right-click the application and click **Run As** > **Run on MobileFirst Development Server**.
- Use the Ant script tool that is described in "Ant tasks for building and deploying applications and adapters" on page 11-77.

The project WAR file is created in the `bin` folder.

5. Configure a database and deploy the project WAR to the application server with one of these two methods:
   - With the MobileFirst Server Configuration Tool. For more information, see "Deploying, updating, undeploying, or upgrading MobileFirst Server by using the Server Configuration Tool" on page 11-11.
   - With Ant tasks for configuring a database for a MobileFirst project and deploying a MobileFirst project WAR file to an application server. With this method, you can also configure the project on the server by 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 11-15.
     - 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 11-16.
     - The list of JNDI environment entries that can be configured is at "JNDI environment entries for MobileFirst projects in production" on page 11-67.
     - You can find sample Ant files that use these Ant tasks in the MobileFirst distribution in *`product_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 15-66.
     a. First call **configuredatabase**, the **`databases`** target in the sample Ant files.
     b. Then call **configureapplicationserver**, the **`install`** target in the sample Ant files.
6. If you want to enable extended authenticity checking, follow the steps in "Configuring extended app authenticity checking" on page 11-56.
7. Open the MobileFirst Operations Console of the target environment.

   If the MobileFirst Operations Console is installed with the default context root, its URL is of the form `https://`*`your-remote-server`*`:`*`server-port`*`/worklightconsole`. If HTTPS is not supported in your application server, it is the unsecured URL `http://`*`your-remote-server`*`:`*`server-port`*`/worklightconsole`.

   Important: If you access the MobileFirst Operations Console through HTTP instead of HTTPS, your MobileFirst administration user password is compromised.
8. From the MobileFirst Operations Console, deploy the relevant `.wlapp` files from the `bin` folder of your MobileFirst project. If you enabled extended authenticity checking in step 6, deploy the `.wlapp` files that resulted from that procedure, instead.
   - For more information about how to deploy an application by using MobileFirst Operations Console, see "Deploying apps" on page 11-85.
   - You can also deploy the app to the target environment by using the MobileFirst Server administration command-line tools. For more information about how to deploy an app by using the provided command-line tools, see

"Administering MobileFirst applications through Ant" on page 12-12 and "Administering MobileFirst applications through the command line" on page 12-37.

9. 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 MobileFirst Operations Console, deploy the .adapter files from the bin folder of your project.
      - For more information about how to deploy an adapter by using MobileFirst Operations Console, see "Deploying adapters" on page 11-86.
      - You can also deploy the adapter to the target environment by using the MobileFirst Server administration command-line tools. For more information about how to deploy an app by using the provided command-line tools, see "Administering MobileFirst applications through Ant" on page 12-12 and "Administering MobileFirst applications through the command line" on page 12-37.

Results

A message is displayed, indicating whether the deployment action succeeded or failed.

Building a project WAR file with Ant

You can build the project WAR file by using Ant tasks.

Before you can run Ant tasks, make sure that Apache Ant is installed. The minimum supported version of Ant is listed in "System requirements" on page 2-9.

Apache Ant 1.8.4 is included in MobileFirst Server. In the *product_install_dir*/shortcuts/ directory, the following scripts are provided:

- For UNIX / Linux: ant
- For Windows: ant.bat

These scripts are ready to run, which means that they do not require specific environment variables. If the JAVA_HOME environment variable is set, the scripts accept it.

Note: Since IBM Worklight Foundation V6.2.0, the worklight-ant-builder.jar file is included in the IBM MobileFirst Platform Command Line Interface for iOS, whereas in earlier versions, it was included in MobileFirst 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.

The Ant task for building a MobileFirst project WAR file has the following structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="myProject" default="all">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="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 MobileFirst runtime environment to start, you must deploy it to the server where the Administration Services application is installed. If you use WebSphere Application Server Network Deployment, you can alternatively install the MobileFirst runtime environment in a server or cluster of the cell other than the one where the Administration Services application that manages this runtime is installed. If you do so, you must start the server or cluster where the Administration Services application is installed before the one where the MobileFirst runtime environment is installed.

Before you start

See “Planning deployment of administration components and runtimes” on page 6-8 for the supported deployment topologies.

Install the MobileFirst Server by following the procedure in “Installing MobileFirst Server” on page 6-3.

If you have a farm topology, configure the server farm by following the procedure in “Installing a server farm” on page 6-127.

To serve users who are using the new session independent mode, as well as existing users whose apps work in session-dependent mode, you must deploy the new V7.1.0 WAR with its V7.1.0 artifacts, but **WITHOUT** removing the old WAR. For more information, see “Upgrading projects to work in session-independent mode” on page 7-10.

The database and application server prerequisites for this task are described in “Installation prerequisites” on page 6-4.

You must build a MobileFirst project WAR file by using MobileFirst Platform Command Line Interface for iOS, or by following the instructions in “Building a project WAR file with Ant” on page 11-4. The WAR file contains the default configuration values for the server, and some resources for the MobileFirst applications and adapters.

- For project WAR files built with earlier versions 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 V6.2.0 and later, and deployed to Worklight Server V6.2.0.1 and later, 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.
- Before deploying WAR files to support session-independent mode, see "Session-independent mode" on page 8-108.

You can deploy a MobileFirst project in one the following ways:
- By using the Server Configuration Tool.
- By using a set of Ant tasks that are supplied with MobileFirst Server to deploy a project WAR file and configure your databases and application servers.
- By creating and configuring the databases manually, and deploying the project WAR file manually.

Optional creation of databases

If you want to activate the option to install the project runtime relational databases when you run the Ant tasks or the Server Configuration Tool, you must have certain database access rights that entitle you to create the databases, or the users, or both, that are required by the project runtime component.

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 must 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.

If you want to use Cloudant for the project runtime database, ensure that you have access to a valid Cloudant account and a user with basic authentication credentials is defined to create the databases. Contrary to relational databases, the installation tools do not create Cloudant databases. They are automatically created when the project runtime is launched on the application server where it is deployed with the Cloudant account URL and user credentials.

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

Important: The manual creation of databases is optional if you install MobileFirst Server 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:

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 11-20 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 name (here `WRKLGHT`) and 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 MobileFirst 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 database:

     ```
     CREATE DATABASE WRKLGHT COLLATE USING SYSTEM PAGESIZE 32768
     CONNECT TO WRKLGHT
     GRANT CONNECT ON DATABASE TO USER wluser
     DISCONNECT WRKLGHT
     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 MobileFirst Platform Foundation for iOS 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 runtime database tables and objects and a SCHEMA for the reports database tables and objects.

Creating the MySQL database:

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 11-28 instead.

About this task

The <configureDatabase> Ant task 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 database for you. When you manually create the database, you can replace the database name (here WRKLGHT) and the password with a database name 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;
   ```
   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 MobileFirst Platform Foundation for iOS runs.

Creating the Oracle database:

You can use the `<configureDatabase>` Ant task to create the Oracle database, except for the Oracle 12c database type.

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 11-32 instead.

About this task

The `<configureDatabase>` Ant task can create the databases, except for the Oracle 12c database type, or the users and schemas inside an existing database 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 might not be supported.

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 MobileFirst 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 tablespace: **`USERS`**
           - Temporary tablespace: **`TEMP`**
           - Status: **`Unlocked`**
           - Add system privilege: **`CREATE SESSION`**
           - Add system privilege: **`CREATE SEQUENCE`**
           - Add system privilege: **`CREATE TABLE`**
           - Add quota: **`Unlimited for tablespace USERS`**
   - Using the `Oracle SQLPlus` command-line interpreter.

     The commands in the following example create a user named *WORKLIGHT*:

```
CONNECT SYSTEM/<SYSTEM_password>@ORCL
CREATE USER WORKLIGHT IDENTIFIED BY 'WORKLIGHT_password' DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS;
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO WORKLIGHT;
DISCONNECT;
```

Creating the Cloudant database:

Before a project runtime can create or access a Cloudant database, you need to configure Cloudant accounts and users. You might also need to specify database prefixes to distinguish projects that have the same name.

General requirements

To use the Cloudant database, you need either an account at Cloudant.com using the Cloudant database in the Cloud, or you need to install Cloudant Local on your own hardware, as described in IBM Cloudant Data Layer Local Edition documentation.

Whether you use Cloudant.com or Cloudant Local, the Cloudant database is accessible via HTTP or HTTPS, and you need the URL to access it and a user name and password.

Cloudant account

Before a project runtime can create or access a Cloudant database at Cloudant.com, you must register a valid account for Cloudant.com. The account registration provides you with the following things:

Account name
The Cloudant account is implicitly also the URL; for example, the account `exampleaccount` corresponds to the URL `https://exampleaccount.cloudant.com`.

User name
By default, the account name serves also as administrator user name of the databases created in your account. Access is granted per database. By default, all databases that you create are only accessible to your account. If you are using a shared database that is accessible to multiple users, you need a user name that has sufficient permissions to write and read documents of the database.

Password
A password to access the account at Cloudant.com.

See Sign Up for a Free Account for information about how to create an account on Cloudant.com.

Cloudant Local

If you use a Cloudant Local installation instead of Cloudant.com, you must configure the following things:

- The URL to access your Cloudant Local installation.
- A user name to access your Cloudant Local installation, which has sufficient permissions to write and read documents in the database.
- A password for this user in your Cloudant Local installation.

For more information about how to create and administer users for Cloudant Local, see Configuring database-level security.

Permissions

The default permissions for a new account at Cloudant.com or at Cloudant Local are sufficient for the project runtime.

The database can be created through the Cloudant dashboard. The user must have at least the following roles set for your database:

_reader
Gives the user permission to read documents from the database.

_writer
Gives the user permission to create and modify documents in the database.

_admin
Gives the user permission to install and update design documents in the database.

Unlike relational databases, there is no concept of “tables” in Cloudant, hence there is no need to create any tables ahead of time or to set any permissions on tables. If the database is not yet created, it will be automatically created when the project runtime is launched on the application server. In this case, the user name must be set to the administrator user name of the Cloudant account, because only the administrator has permission to create new databases.

Default database name for project runtimes

Each deployed project needs to have a separate database. By default, the name of this database is in the following form:

```
{project_name}_runtime_db
```

where `{project_name}` is the context root of the project without the leading forward slash.

If multiple teams use the same database with projects that have the same names, they can specify a prefix through a "**`dBNamePrefix`**" at deployment time to distinguish each project either manually or with Ant tasks. For more information about how to use this prefix manually for the application server to which you plan to deploy your MobileFirst runtime, see the relevant subsection in “Deploying a project WAR file and configuring the application server manually” on page 11-37. For more information about how to use this prefix with Ant tasks, see “Ant tasks for installation of MobileFirst runtime environments” on page 15-32, Table 15-60 on page 15-44.

Note:

Databases for projects are created if needed after deployment when the project is launched.

Deploying, updating, undeploying, or upgrading MobileFirst Server by using the Server Configuration Tool

The Server Configuration Tool is a graphical tool that you can use to deploy, update, undeploy, or upgrade a MobileFirst Server instance to or from an application server and database.

If you use this tool in production to upgrade a MobileFirst Server, you must complete more actions to upgrade the server, as described in “Upgrading to MobileFirst Server V7.1.0 in a production environment” on page 7-13.

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 6-5.

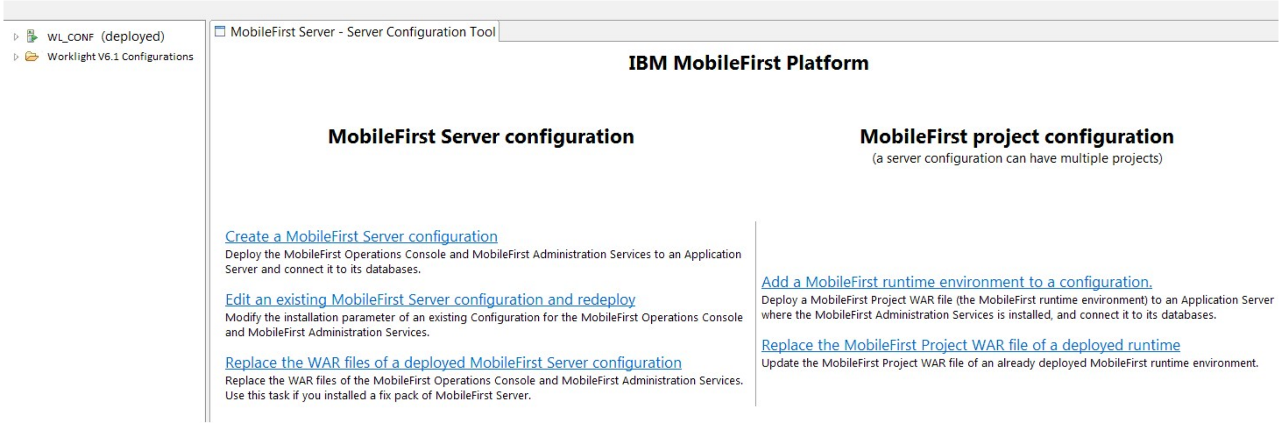
The Server Configuration Tool provides the same capabilities as the Ant tasks that are described in “Ant tasks for installation of MobileFirst Operations Console and Administration Services” on page 15-24 and “Ant tasks for installation of MobileFirst runtime environments” on page 15-32. Compared to Ant tasks, the Server Configuration Tool is limited to a set of operations that are described in the following list:

- The supported databases are IBM DB2, Oracle, MySQL, and Cloudant.

  Restriction: The Derby database is not supported.
- It is not possible to define JNDI deployment properties, such as **`publicWorkLightHostname`** or other properties that are listed in “JNDI environment entries for MobileFirst projects in production” on page 11-67. 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 11-14.)

- 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 MobileFirst 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 MobileFirst Server is not supported for production in this environment.
- You can use the Server Configuration Tool to install MobileFirst 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 MobileFirst Server to a server farm.
- You cannot use the Server Configuration Tool to add a runtime to an Administration Service that was installed or upgraded with Ant tasks and not with the Server Configuration Tool.



WL_CONF (deployed)

Worklight V6.1 Configurations

MobileFirst Server - Server Configuration Tool

IBM MobileFirst Platform

MobileFirst Server configuration

Create a MobileFirst Server configuration

Deploy the MobileFirst Operations Console and MobileFirst Administration Services to an Application Server and connect it to its databases.

Edit an existing MobileFirst Server configuration and redeploy

Modify the installation parameter of an existing Configuration for the MobileFirst Operations Console and MobileFirst Administration Services.

Replace the WAR files of a deployed MobileFirst Server configuration

Replace the WAR files of the MobileFirst Operations Console and MobileFirst Administration Services. Use this task if you installed a fix pack of MobileFirst Server.

MobileFirst project configuration

(a server configuration can have multiple projects)

Add a MobileFirst runtime environment to a configuration.

Deploy a MobileFirst Project WAR file (the MobileFirst runtime environment) to an Application Server where the MobileFirst Administration Services is installed, and connect it to its databases.

Replace the MobileFirst Project WAR file of a deployed runtime

Update the MobileFirst Project WAR file of an already deployed MobileFirst runtime environment.

Figure 11-1. 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 *mf_server_install_dir*/shortcuts/configuration-tool.sh.

  Note: *mf_server_install_dir* is the directory where you install MobileFirst Server. *mf_server* is the shortcut for MobileFirst Server.
- From a shell command line, run the command *mf_server_install_dir*/shortcuts/configuration-tool.sh.

On Windows

- By using the **Start** > **IBM MobileFirst Platform Server** > **Server Configuration Tool** menu command.

- In Windows Explorer, double-click the file *mf_server_install_dir*/shortcuts/configuration-tool.bat.
- In a console window, run *mf_server_install_dir*/shortcuts/configuration-tool.bat.

On Mac OS X

Restriction: MobileFirst Server is not supported for production in this environment.

- In the Finder, double-click the file *mf_server_install_dir*/shortcuts/configuration-tool.sh.
- In a Terminal window, run *mf_server_install_dir*/shortcuts/configuration-tool.sh.

Main tasks

Create a MobileFirst Server configuration

For more information, see "Installing MobileFirst Server administration with the Server Configuration Tool" on page 6-60.

After a MobileFirst Server configuration is created, you can do the following tasks:

Edit an existing MobileFirst Server configuration and redeploy

Use this task to edit and modify an existing MobileFirst 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 MobileFirst administration components are uninstalled from the application server, and reinstalled with the new parameters.

Add a MobileFirst runtime environment to a configuration

Use this task to add a MobileFirst runtime environment to a MobileFirst Server configuration.

To create a new MobileFirst runtime environment, complete the following steps:

1. Select a MobileFirst Server configuration.
2. Select **File** > **Add MobileFirst runtime environment**.
3. Enter a descriptive name for the MobileFirst runtime environment.
4. Select the path for the MobileFirst project WAR file to be deployed.
5. Step through the wizard to describe the target database management system.

   If you need to create a relational database for your MobileFirst 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 MobileFirst 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" on page 11-6.

Note: The Server Configuration Tool does not create a database if you select Cloudant as the database management system for your MobileFirst runtime environment. The database name depends on values that you are prompted for, and is displayed on the **Database Creation Request** panel.

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 MobileFirst tables, these tables are created.
   c. If the database contains MobileFirst tables for an older version of the product, the tables are upgraded to the current version.
   d. If the database operations succeed, the MobileFirst 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
Use this task to update the WAR files and libraries of the MobileFirst administration components. For example, apply a fix pack to the installation directory of MobileFirst Server.

Replace the WAR file of a deployed MobileFirst Server configuration
If you applied a fix pack to your installation of MobileFirst 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 MobileFirst Operations Console and Administration Services, and for each MobileFirst 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 MobileFirst Server operations in batch mode (from the command line and without using a graphical user interface).

If you modify the MobileFirst Server status outside the Server Configuration Tool, the status for this configuration is no longer accurate.

Migrate a V6.1.0 Configuration
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 MobileFirst Platform Foundation for iOS V7.1.0, 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 MobileFirst Platform Foundation for iOS V7.1.0 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 IBM Worklight runtime environment V6.1.0 is removed from the application server. The databases are migrated, the MobileFirst Operations Console and Administration Services are deployed, and the MobileFirst runtime environment is deployed.

Upgrade a MobileFirst configuration to the current version with the Server Configuration Tool

Since IBM MobileFirst Platform Foundation for iOS V7.1.0, you can use the Server Configuration Tool to upgrade an older configuration already deployed, in a single action. For more information, see “Upgrading with the Server Configuration Tool” on page 7-72.

Change the working directory where the configurations are stored

Click **File** > **Preferences** and select a different working directory.

Using Ant tasks to deploy the project WAR file

Creating and configuring the databases with Ant tasks:

If you did not manually create databases, you can use Ant tasks to create and configure your databases.

About this task

If you did not use the procedure in “Optional creation of databases” on page 11-6 to create the databases manually, complete the following steps.

Important: You use this procedure to create a relational database, if needed, and to populate it with the required tables. If you deploy with Cloudant as a database, you do not need to perform this procedure.

Procedure

1. Review the sample configuration files in “Sample configuration files” on page 15-66, and copy the Ant file that corresponds to your database. The files for creating a database are named after the following pattern:

   ```
   create-database-database.xml
   ```

   For more information, see table 1, Table 15-83 on page 15-67, in “Sample configuration files” on page 15-66.
2. Follow step 4 of the page “Sample configuration files” on page 15-66 to edit the Ant file and replace the placeholder values for the properties at the beginning of the file.
3. Run the following commands to create the databases.

   ```
   ant -f create-database-database.xml databases
   ```

You can find the Ant command in *product_install_dir*/shortcuts.

If the databases are created, and you must create only the databaseTABLES.

4. Edit the Ant script that you use later to create and configure the databases.
5. Review the sample configuration files in "Sample configuration files" on page 15-66, and copy the Ant file that corresponds to your database. The files for configuring an existing database are named after this pattern:

   ```
   configure-appServer-database.xml
   ```

   For more information, see table 1, Table 15-83 on page 15-67, in "Sample configuration files" on page 15-66.
6. See step 4 of the page "Sample configuration files" on page 15-66 to edit the Ant file and replace the placeholder values for the properties at the beginning of the file.
7. Run the following commands to create the databases.

   ```
   ant -f configure-appServer-database.xml databases
   ```

   You can find the Ant command in *product_install_dir*/shortcuts.

What to do next

See also:
- "Ant **configuredatabase** task reference" on page 15-14
- "Sample configuration files" on page 15-66

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 that are used by IBM MobileFirst Platform Foundation for iOS. A set of Ant tasks is supplied with IBM MobileFirst Platform Server.

Before you begin

See "Planning deployment of administration components and runtimes" on page 6-8 for the supported deployment topologies.

Before you deploy your project WAR file and configure the application server, you must complete the following procedures:
- "Installing MobileFirst Server" on page 6-3.
- "Installing a server farm" on page 6-127, to configure the server farm if you have a farm topology.
- "Creating and configuring the databases with Ant tasks" on page 11-15 (only for relational databases)

If you use Cloudant via https as a database, and WebSphere Application Server full profile as an application server, declare the Cloudant signer certificate, as described in "Configuring WebSphere Application Server for use with Cloudant" on page 6-57.

If you deploy different project WAR files with relational databases, each WAR file must have its own set of tables. Likewise, if you deploy different project WAR files with Cloudant, each WAR file must have its own database to store the JSON documents. Each supported database type has the following guidelines:

- For DB2, each WAR file must have different databases, or schema to store the tables.
- For MySQL, each WAR file must have different databases to store the tables.
- For Oracle, each WAR file must have different database users to store the tables.
- For Derby, each WAR file must have different databases to store the tables.
- For Cloudant, each WAR file must have different databases, unless a prefix is specified for the database name, so that several teams use the same database. For more information, see "Creating the Cloudant database" on page 11-9.

You must run the Ant task on the computer where the application server is installed, or the Network Deployment Manager for WebSphere Application Server Network Deployment. If you want to start the Ant task from a computer on which Worklight Server is not installed, you must copy the file `*mf_server_install_dir*/WorklightServer/worklight-ant-deployer.jar` to that computer.

The *mf_server_install_dir* placeholder is the directory where you installed IBM MobileFirst Platform Server.

Note: If your application server is WebSphere Application Server V7.0, you must add the configuration property "`com.ibm.ws.webcontainer.invokerequestlistenerforfilter = true`". For more information about how to add this property, see "Configuring WebSphere Application Server V7.0" on page 6-59.

Procedure

1. Review the environment ID that you used to install the MobileFirst Server administration. This environment ID is installed as a JNDI property. For more information about the list of JNDI properties, see "List of JNDI properties for MobileFirst Server administration" on page 6-101.

   Important: If the MobileFirst Server administration used an environment ID, install the project WAR file with the same environment ID. Otherwise, that project WAR file cannot be managed by the MobileFirst Server administration.
2. Edit the Ant script that you use later to deploy the Project WAR File.
   a. Review the sample configuration files in "Sample configuration files" on page 15-66, and copy the Ant file that corresponds to your database. The files for deploying a project WAR file are named after the following pattern:

      `configure-*appServer*-*database*.xml`

      For more information, see table 1, Table 15-83 on page 15-67, in "Sample configuration files" on page 15-66.

      Note: If your file name follows the pattern `configure-*appServer*-*database*.xml`, you can reuse it for "Creating and configuring the databases with Ant tasks" on page 11-15,
   b. Follow step 4 of the page "Sample configuration files" on page 15-66 to edit the Ant file and replace the placeholder values for the properties at the beginning of the file.
   c. Each project WAR file that uses Cloudant as a database must have its own database. By default, the database includes the project name. If several teams use the same Cloudant database with projects that have the same

names, make the distinction by using the `dbNamePrefix` of the `<cloudant>` element in the Ant task **configureapplicationserver**.

3. If the MobileFirst Server administration uses an environment ID, and you run an Ant task for the installation, add an `environmentID` attribute to the following Ant tasks, which are used for the administration installation:
   - **installworklightadmin**
   - **updateworklightadmin**
   - **uninstallworklightadmin**
   - **configureapplicationserver**
   - **updateapplicationserver**
   - **unconfigureapplicationserver**

   For more information, see "Ant tasks for installation of MobileFirst runtime environments" on page 15-32 and "Ant tasks for installation of MobileFirst Operations Console and Administration Services" on page 15-24.

   Important: The value of the attribute must be the same as the one used for the MobileFirst Server administration.
4. To deploy the project WAR file, run the following command:
   ```
   ant -f configure-appServer-database.xml install
   ```
   You can find the Ant command in *mf_server_install_dir*/shortcuts
5. Save the Ant file. You might need it later to apply a fix pack or perform an upgrade. For more information, see "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21. If you do not want to save the passwords, you can replace them by "************" (12 stars) for interactive prompting.

What to do next

See also:
- "Ant tasks for installation of MobileFirst runtime environments" on page 15-32
- "Sample configuration files" on page 15-66
- "Using Ant tasks to install MobileFirst Server administration" on page 6-62
- "Encrypting database password with Ant tasks for Liberty" on page 15-22

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 MobileFirst 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Runtime Component to be installed. The following table lists the available elements:

   Note: You must choose the same inner element to install the MobileFirst Administration Services. You must install an instance of the MobileFirst Administration Service on each server where the MobileFirst Runtime Component is installed.

Table 11-1. Inner elements of `<was>` for network deployment

Element	Explanation
cell	Install the MobileFirst project into all application servers of the cell.
cluster	Install the MobileFirst project into all application servers of the specified cluster.
node	Install the MobileFirst project into all application servers of the specified node that are not in a cluster.
server	Install the MobileFirst 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 MobileFirst 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 MobileFirst Server in different configurations or even different versions of MobileFirst 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 MobileFirst 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 11-2. 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 the project WAR file manually

Creating and configuring the databases manually:

You can manually create and configure the IBM MobileFirst Platform Foundation for iOS databases.

Note: The reports database and the sample BIRT Reports are deprecated since IBM MobileFirst Platform Foundation for iOS V7.0.0. However, if you need it, you can still manually create and configure the reports database. For more information about this procedure, see "Manually creating and configuring the reports database" on page 13-117.

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 database" on page 11-6.
2. Create the tables in the databases. This step is described in "Setting up your DB2 database 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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

Setting up your DB2 database manually:

You can set up the database manually instead of using Ant tasks.

About this task

Set up your DB2 database by creating the database schema. The following procedure creates the schemas for WRKLGHT.

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 instances of IBM MobileFirst Platform Server 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 product_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.

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.

Before you begin

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` can 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 runtime 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>
```

   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 **`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 IBM MobileFirst Platform Server runtime operations” on page 6-20.

   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 MobileFirst Server application, following the instructions in “Configuring the Liberty profile manually” on page 11-38. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"`.
3. You can encrypt the database password with the securityUtility program in `<liberty_install_dir>/bin`.

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 to the directory that you determined in step 1.

   You can retrieve the file in one of two ways:
   - Download it from DB2 JDBC Driver Versions.
   - Fetch it from the *db2_install_dir*/java on the DB2 server directory.
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 runtime 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 **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 on page 11-23 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. Test the data source connection by selecting each **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:

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 to `$TOMCAT_HOME/lib`.

   You can retrieve the file in one of two ways:

   - Download it from DB2 JDBC Driver Versions
   - Fetch it from the *`db2_install_dir`*`/java` directory on the DB2 server.
2. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the `server.xml` file, as indicated in “Configuring Apache Tomcat manually” on page 11-46.

```
<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;"/>
```

Where **worklight** after **username=** is the name of the system user with "CONNECT" access to the **WRKLGHT** database 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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

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 database 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 11-26
   - "Configuring WebSphere Application Server for Derby manually" on page 11-26
   - "Configuring Apache Tomcat for Derby manually" on page 11-28

Setting up your Apache Derby database manually:

You can set up your Apache Derby database manually instead of by running Ant tasks.

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.

   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.

   For supported versions of Apache Derby, see "System requirements" on page 2-9.
   The script displays ij version number.
2. At the command prompt, enter the following commands:

```
connect 'jdbc:derby:WRKLGHT;user=WORKLIGHT;create=true';
run 'product_install_dir/WorklightServer/databases/create-worklight-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 runtime database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false" statementCacheSize="10">
  <jdbcDriver libraryRef="derbyLib"
              javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource40"/>
  <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>
```

The `jndiName` attributes must depend on the context root that you select for the MobileFirst Server application, following the instructions in “Configuring the Liberty profile manually” on page 11-38. If the context root is `/app_context`, use `jndiName="app_context/jdbc/WorklightDS"`.

Configuring WebSphere Application Server for Derby manually:

You can set up and configure your Apache Derby database manually with WebSphere Application Server.

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 *product_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 runtime 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** data source 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** data source that you created.
x. Click **test connection** (only if you are not on the console of a WAS 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:

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 *product_install_dir*/ApplicationCenter/tools/lib/derby.jar to the directory $TOMCAT_HOME/lib.
2. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the server.xml file, as indicated in "Configuring Apache Tomcat manually" on page 11-46.

```
<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"/>
```

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 database" on page 11-7.
2. Create the tables in the databases. This step is described in "Setting up your MySQL database manually."
3. Perform the application server-specific setup as the following list shows.

Setting up your MySQL database 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 'worklight';
GRANT ALL PRIVILEGES ON WRKLGHT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
FLUSH PRIVILEGES;

USE WRKLGHT;
SOURCE product_install_dir/WorklightServer/databases/create-worklight-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 MobileFirst Platform Foundation for iOS 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.
3. Add the following property to your MySQL option file: `innodb_log_file_size = 250M`

   For more information about `innodb_log_file_size`, see the MySQL documentation, section innodb_log_file_size.

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

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.

Before you begin

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 runtime database -->
<dataSource jndiName="worklight/jdbc/WorklightDS" transactional="false">
  <jdbcDriver libraryRef="MySQLLib"/>
  <properties databaseName="WRKLGHT"
              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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

The jndiName attributes must depend on the context root that you select for the MobileFirst Server application, following the instructions in "Configuring the Liberty profile manually" on page 11-38. If the context root is /app_context, use jndiName="app_context/jdbc/WorklightDS".

3. You can encrypt the database password with the securityUtility program in <liberty_install_dir>/bin.

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 runtime 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. 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
      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 IBM MobileFirst Platform Server runtime operations" on page 6-20.
6. 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**.

7. 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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the `server.xml` file, as indicated in "Configuring Apache Tomcat manually" on page 11-46.

   ```
   <Resource name="jdbc/WorklightDS"
             auth="Container"
             type="javax.sql.DataSource"
             maxActive="100"
             maxIdle="30"
             maxWait="10000"
             username="worklight"
             password="worklight"
             driverClassName="com.mysql.jdbc.Driver"
             url="jdbc:mysql://mysqlserver:3306/WRKLGHT"/>
   ```

   Where **`worklight`** after **`username=`** is the user name of the MySQL server, **`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).

   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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

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 database" on page 11-8.
2. Create the tables in the databases. This step is described in "Setting up your Oracle database manually."
3. Perform the application server-specific setup as the following list shows.

Setting up your Oracle database manually:

You can set up the database manually instead of using 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, set the character set of the database to **Unicode (AL32UTF8)**.

   If the Oracle installation is on a UNIX or Linux computer, make sure that the database is started the next time the Oracle installation is restarted. To this effect, make sure that 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.
   - To create the user for the runtime database/schema (by default Worklight), by using Oracle Database Control, proceed as follows:
     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: Unlocked
     Add system privilege: CREATE SESSION
     Add system privilege: CREATE SEQUENCE
     Add system privilege: CREATE TABLE
     Add quota: Unlimited for tablespace USERS
     ```
   - To create the user by using Oracle SQLPlus, enter the following commands:

```
CONNECT SYSTEM/SYSTEM_password@ORCL
CREATE USER WORKLIGHT IDENTIFIED BY WORKLIGHT_password DEFAULT TABLESPACE USERS QUOTA UNLIMITED ON USERS;
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO WORKLIGHT;
DISCONNECT;
```

3. Create the database tables for the runtime database:
   a. Using the Oracle SQLPlus command-line interpreter, create the tables for the runtime database by running the create-worklight-oracle.sql file:

```
CONNECT WORKLIGHT/product_password@ORCL
@product_install_dir/WorklightServer/databases/create-worklight-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.

Before you begin

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 runtime 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>
```

   Where **WORKLIGHT** after **user=** is the name of the user with "CONNECT" access to the **WRKLGHT** database 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 **oserver** with the host name of your Oracle server (for example, localhost, if it is on the same machine).

   Note:

   - For more information on how to connect the Liberty server to the Oracle database with a service name, or with a URL, see the WebSphere Application Server Liberty Core 8.5.5 documentation, section **properties.oracle**.
   - 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 IBM MobileFirst Platform Server runtime operations” on page 6-20.

   The jndiName attributes must depend on the context root that you select for the MobileFirst Server application, following the instructions in “Configuring the Liberty profile manually” on page 11-38. If the context root is /app_context, use jndiName="app_context/jdbc/WorklightDS".
3. You can encrypt the database password with the securityUtility program in <liberty_install_dir>/bin.

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 11-3. 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 runtime 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. For WebSphere Application Server Network Deployment, click **System administration** > **Nodes**, select the nodes, and click **Full Synchronize**.

What to do next

Note: The database user **WORKLIGHT** 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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

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. Prepare an XML statement that defines the data source, as shown in the following code example. Insert this statement in the server.xml file, as indicated in "Configuring Apache Tomcat manually" on page 11-46.

```
<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"/>
```

Where **WORKLIGHT** after **username=** is the name of the user with "CONNECT" access to the **WRKLGHT** database 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 `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 IBM MobileFirst Platform Server runtime operations" on page 6-20.

Configuring the Cloudant databases manually:

Use JNDI properties to configure the Cloudant databases.

Procedure

Configure the Cloudant database by setting JNDI properties in the relevant application server.
No manual configuration is necessary from the Cloudant perspective. If the database for the project runtime does not exist, and if the Cloudant user that is provided to the project runtime is the Cloudant administrator, then the database is automatically created when the MobileFirst project environment is started. If the user that is provided to the project runtime is not allowed to create a database, you must create a database with the names that are documented at "Creating the Cloudant database" on page 11-9.

What to do next

For more information about MobileFirst runtime configuration with Cloudant, see "Deploying a project WAR file and configuring the application server manually."

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 the product that was used to build the project WAR file and the version of MobileFirst Server, you might need to migrate the WAR file first.

When the version of IBM MobileFirst Platform Foundation for iOS produces a WAR file that is not compatible with the version of MobileFirst Server, you must migrate the project WAR file to the current MobileFirst Server version to ensure a successful manual deployment. All fix packs of a product version are compatible in that sense, and so you do not need to migrate associated project WAR files.

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 MobileFirst Server:

Use Ant tasks to migrate a project WAR file so that you can deploy it manually to a new version of MobileFirst 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 V5.0.6 of this product 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/myMigratedProject.war"/>
  </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 Liberty profile manually:

To configure the WebSphere Application Server Liberty profile manually, you must modify the `server.xml` file and declarations for the runtime and the IBM MobileFirst Platform Operations Console.

Before you begin

See “Planning deployment of administration components and runtimes” on page 6-8 for the supported deployment topologies.

Review the environment IDs. Environment IDs are optional, but if you do specify one, the identifier must meet the following two conditions:

- Its value must be the same for each MobileFirst runtime environment that is managed by the same MobileFirst Server administration component.
- Its value must match the environment ID that is used when the MobileFirst 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 7 on page 11-40.

About this task

In addition to modifications for the databases that are described in “Creating and configuring the databases manually” on page 11-20, you must make the following modifications to the `server.xml` file.

Note: In the following procedure, when the example uses the `worklight.war` project file, use the name of your MobileFirst 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, 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 MobileFirst 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 product_install_dir/WorklightServer/worklight-jee-library.jar WLP_DIR/usr/shared/resources/lib
```

   - On Windows:

```
mkdir WLP_DIR\usr\shared\resources\lib
copy /B product_install_dir\WorklightServer\worklight-jee-library.jar WLP_DIR\usr\shared\resources\
lib\worklight-jee-library.jar
```

3. Ensure that the `<featureManager>` element contains at least the following `<feature>` elements:
   - For WebSphere Application Server Liberty profile V8.5.0.x:

```
<feature>ssl-1.0</feature>
<feature>servlet-3.0</feature>
<feature>jdbc-4.0</feature>
```

   - For WebSphere Application Server Liberty profile V8.5.5.0 and later:

```
<feature>jdbc-4.0</feature>
<feature>servlet-3.0</feature>
<feature>ssl-1.0</feature>
```

4. Copy the `worklight.war` file to the apps directory of the Liberty server.

   Note: The apps directory is in the same directory as the `server.xml` file.
5. Add the following declarations in the `<server>` element for the MobileFirst runtime.

   Important:
   - The **`id`** attribute of the **`privateLibrary`** tag must identify a unique MobileFirst runtime. By convention, it takes this form: `<privateLibrary id="worklightlib_`*`<context root>`*`">`
   - The version number of the `com.ibm.ws.crypto.passwordutil_*.jar` cryptographic JAR file might change according to the version of WebSphere Application Server Liberty profile or its fix packs.

```
<!-- Declare the MobileFirst Server application. -->
<application id="worklight" name="worklight" location="worklight.war" type="war">
  <classloader delegation="parentLast">
    <privateLibrary id="worklightlib_worklight">
      <fileset dir="${shared.resource.dir}/lib" includes="worklight-jee-library.jar"/>
      <fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil_*.jar"/>
    </privateLibrary>
  </classloader>
</application>

<!-- Declare web container custom properties for the MobileFirst Server application. -->
<webContainer invokeFlushAfterService="false"/>
```

This declaration installs the MobileFirst 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" type="war">
```

In either case, also change the **privateLibrary** tag accordingly: <privateLibrary id="worklightlib_app_context">

6. If the database is Oracle, add the **commonLibraryRef** attribute to the class loader of the worklight runtime application.

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

The name of the library reference (OracleLib in this example) must be the ID of the library that contains the JDBC JAR file. This ID is declared in the procedure that is documented in "Configuring Liberty profile for Oracle manually for MobileFirst Server administration" on page 6-78.

7. If the MobileFirst Server administration component uses an environment ID, declare that environment ID for MobileFirst Server application:

```
<jndiEntry jndiName="worklight/ibm.worklight.admin.environmentid" value='"ValueOfEnvironmentID"'/>
```

- worklight is the context root of the MobileFirst Server application. If you choose another value in previous step, replace worklight with that value.
- Replace ValueOfEnvironmentID with the value that is used for the MobileFirst Server administration component.

8. If you want to use Cloudant as the database for the MobileFirst runtime you defined in step 5 on page 11-39, then add the following JNDI properties:

```
<!-- Declare the JNDI properties for the IBM Worklight project runtime for Cloudant. -->
<jndiEntry jndiName="worklight/mfp.db.cloudant.url" value='"https://cloudant.com:443"'/>
<jndiEntry jndiName="worklight/mfp.db.cloudant.username" value='"wluser"'/>
<jndiEntry jndiName="worklight/mfp.db.cloudant.password" value='"password"'/>
```

- worklight is the context root of the MobileFirst Server application. If you choose another value in previous step, replace worklight with that value.
- Replace https://cloudant.com:443 with the URL to connect to Cloudant.
- Replace wluser with the name of the user that has access the MobileFirst runtime database. If this user is not allowed to create a database, you must create a database manually, with the names that are documented in "Creating the Cloudant database" on page 11-9.
- Replace password with the password associated with this user.

Note: Each project using Cloudant as a database needs its own database. By default, the database includes the project name. But if several teams use the same Cloudant database with projects that have same names, add JNDI property **mfp.db.cloudant.dbNamePrefix** to distinguish the projects as indicated below, and replace *your_prefix* with a value of your choosing.

```
<jndiEntry jndiName="worklight/mfp.db.cloudant.dbNamePrefix" value='"your_prefix"'/>
```

Important: The Cloudant database name must start with a lowercase letter, and can contain only lowercase characters (a-z), digits (0-9), and any of the following characters: _, $, -.

The value that you specify for the variable *mfp.db.cloudant.dbNamePrefix* must comply with this rule.

9. If you installed MobileFirst Operational Analytics, and you want it to collect data from the MobileFirst runtime, add the following JNDI properties:

```
<jndiEntry jndiName ="worklight/wl.analytics.url" value ='"http://analytics.host:port/analytics-service/data"'/>
<jndiEntry jndiName ="worklight/wl.analytics.console.url" value ='"http://analytics.host:port/analytics/console"'/>
<jndiEntry jndiName ="worklight/wl.analytics.username" value ='"wlanalyticsuser"'/>
<jndiEntry jndiName ="worklight/wl.analytics.password" value ='"wlanalyticspassword"'/>
```

- `worklight` is the context root of the MobileFirst Server application. If you select another value in the previous step, replace `worklight` with that value.
- Replace `analytics.host:port` with the host name and port of the server where you installed MobileFirst Operational Analytics.
- Replace `wlanalyticsuser` with the name of the user who has access to MobileFirst Operational Analytics.
- Replace `wlanalyticspassword` with the password that is associated with this user.

Note: The user name and password that you provide in replacement of `wlanalyticsuser` and `wlanalyticspassword` must match the credentials of the user that you defined in the server where MobileFirst Operational Analytics is installed.

Related reference:

"JNDI environment entries for MobileFirst projects in production" on page 11-67
JNDI environment entries cover all the properties that you can set in a production environment. Set production environment properties by configuring your project WAR file with JNDI environment entries.

Related information:

WebSphere Application Server documentation about Deploying application to the Liberty Profile

Configuring WebSphere Application Server manually:

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

Before you begin

See "Planning deployment of administration components and runtimes" on page 6-8 for the supported deployment topologies.

If you use Cloudant via https as a database, and WebSphere Application Server full profile as an application server, declare the Cloudant signer certificate, as described in "Configuring WebSphere Application Server for use with Cloudant" on page 6-57.

Find the SOAP port of the deployment manager (WebSphere Application Server Network Deployment only)

These instructions assume that a stand-alone profile exists 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 MobileFirst Administration Services.

Review the environment IDs

Specifying an environment ID is optional. However, if you specify an ID, use the same value for each MobileFirst runtime environment that is managed by the same MobileFirst Server administration component. Moreover, this value must match the environment ID that is used when the MobileFirst Server administration component is installed. For more information about the **`ibm.worklight.admin.environmentid`** JNDI property, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

Note: If you use WebSphere Application Server V7.0, you must add the configuration property "`com.ibm.ws.webcontainer.invokerequestlistenerforfilter = true`". For more information about how to add this property, see "Configuring WebSphere Application Server V7.0" on page 6-59.

Procedure

1. Determine a suitable file name for the MobileFirst 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 *`product_install_dir`*`/WorklightServer/worklight-jee-library.jar` to the location that you determined in step 1.
3. Log on to theWebSphere Application Server administration console for your MobileFirst Server.

   The address is of the form `http://`*`server`*`.com:9060/ibm/console`, where *server* is the name of the server.
4. Create the MobileFirst 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 MobileFirst JDBC data source and provider.

   Note: You do not need to perform this step if you use Cloudant as a database.

   See the instructions for the appropriate DBMS in “Creating and configuring the databases manually” on page 11-20.

6. Add a specific web container custom property.

   a. Click **Servers** > **Server Types** > **Application Servers**, and select the server for IBM MobileFirst Platform Foundation for iOS.

   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 11-4. Values for the web container custom property.

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 MobileFirst project WAR file.

   Note: In the following procedure, when the example uses `worklight.war`, use the name of your MobileFirst 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 MobileFirst Server installation directory *`product_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 resource references to resources" page.

k. If you use a relational database, enter the JNDI name you created in step 5 on page 11-43 for the `jdbc/WorklightDS` data source.

l. You must also enter a default data source in the following cases:

   - You do not use the reports database, whose resource reference is `jdbc/WorklightReportsDS`. The reports database is not used by default. For more information, see "(deprecated) Reports database" on page 13-94.
   - You do not need any relational database because you use Cloudant. The data source is not used in this case.

   To enter a default data source, enter `DefaultDatasource` in the **Target Resource JNDI Name** field. If the `DefaultDatasource` does not exist in your application server, a message displays **Application Resource Warning** when you quit the page. Ignore the warning and click **Continue**.

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

n. In the **Context Root** field, type `/worklight`.

o. Click **Next**.

p. In **Map environment Entries for Web Module**, you can assign the JNDI variables according to your configuration.

   - Set the variable *`ibm.worklight.topology.platform`* to `WAS`
   - Set 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, set the value of *`ibm.worklight.topology.clustermode`* to `Standalone`
   - On WebSphere Application Server Network Deployment, set the variables as follows:
     - *`ibm.worklight.topology.clustermode`*: `Cluster`
     - *`ibm.worklight.admin.jmx.dmgr.host`*: the host name of the deployment manager
     - *`ibm.worklight.admin.jmx.dmgr.port`*: the SOAP port of the deployment manager
   - If you want to use Cloudant as the database for the MobileFirst runtime, set the variables as follows:
     - `mfp.db.cloudant.url`: the URL to connect to Cloudant
     - `mfp.db.cloudant.username`: the Cloudant account user name
     - `mfp.db.cloudant.password`: the Cloudant account password

   Notes:

   - If this user is not allowed to create a database, you must create a database manually with the names that are documented in "Creating the Cloudant database" on page 11-9.
   - Each project that uses Cloudant as a database must have its own database. By default, the database includes the project name. But if several teams use the same Cloudant database with projects that have same names, add the JNDI property **`mfp.db.cloudant.dbNamePrefix`** to distinguish the projects.

   Important: The Cloudant database name must start with a lowercase letter, and can contain only lowercase characters (a-z), digits (0-9), and any of the following characters: `_`, `$`, `-`.

The value that you specify for the variable *mfp.db.cloudant.dbNamePrefix* must comply with this rule.

- If you installed MobileFirst Operational Analytics, and you want it to collect data from the MobileFirst runtime, set the variables as follows:
  - wl.analytics.url is the URL to connect to a server where MobileFirst Operational Analytics Services is installed.
  - wl.analytics.console.url is the URL to connect to a server where MobileFirst Operational Analytics Console is installed.
  - wl.analytics.username is the user name used for a basic authentication to access the URL that you entered in wl.analytics.url.
  - wl.analytics.password is the password for the user that you entered in wl.analytics.username.

Note: The user name and password that you provide in wl.analytics.username and wl.analytics.password must match the credentials of the user that you defined in the server where MobileFirst Operational Analytics is installed.

q. 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 MobileFirst Administration service must already be available when the MobileFirst 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 MobileFirst 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 MobileFirst 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 MobileFirst Administration Console that is installed in "Installing the MobileFirst Server administration" on page 6-47. The default URL of the MobileFirst Administration Console is `http://<server>:<port>/worklightconsole`, where *server* is the host name of your server and *port* is the port number (default value `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

See "Planning deployment of administration components and runtimes" on page 6-8 for the supported deployment topologies.

Review the environment IDs. Specifying an environment ID is optional. However, if you speciy an ID, use the same value for each MobileFirst runtime environment that is managed by the same MobileFirst Server administration component. Moreover, this value must match the environment ID that is used when the MobileFirst Server administration component is installed. For more information about the **ibm.worklight.admin.environmentid** JNDI property, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

Procedure

1. Copy the MobileFirst JAR file to the Tomcat `lib` directory:
    - On UNIX and Linux systems: `cp product_install_dir/WorklightServer/worklight-jee-library.jar tomcat_install_dir/lib`
    - On Windows systems: `copy /B product_install_dir\WorklightServer\worklight-jee-library.jar tomcat_install_dir\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 11-20.

3. Copy the MobileFirst project WAR file to the Tomcat web application directory, *tomcat_install_dir*/webapps, and rename it according to the 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_install_dir*/conf/server.xml to declare the context and properties of the MobileFirst application:

```
<!-- Declare the MobileFirst runtime environment. -->
<Context path="/worklight" docBase="worklight">
     <Environment name="ibm.worklight.topology.platform" value="Tomcat" type="java.lang.String" override="false"/>
     <Environment name="ibm.worklight.topology.clusterMode" value="Standalone" type="java.lang.String" override="false"/>
      <!-- Declare the worklight and worklight reports databases. -->
      <!-- <Resource name="jdbc/WorklightDS" type="javax.sql.DataSource" ... /> -->
      <!-- <Resource name="jdbc/WorklightReportsDS" type="javax.sql.DataSource" ... /> -->
</Context>
```

   Where you must uncomment and complete the <Resource> element to declare the administration database as described in the following sections:
   - “Configuring Apache Tomcat for DB2 manually” on page 11-24
   - “Configuring Apache Tomcat for Derby manually” on page 11-28
   - “Configuring Apache Tomcat for MySQL manually” on page 11-32
   - “Configuring Apache Tomcat for Oracle manually” on page 11-36

   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. If you want to use Cloudant as the database for the MobileFirst runtime that you defined in step 4, add the following JNDI properties inside the <Context path="/worklight" docBase="worklight"> element.

```
<Environment name="https://cloudant.com:443" type="java.lang.String" override="false"/>
<Environment name="mfp.db.cloudant.username" value="wluser" type="java.lang.String" override="false"/>
<Environment name="mfp.db.cloudant.password" value="password" type="java.lang.String" override="false"/>
```

   - Replace https://cloudant.com:443 with the URL to connect to Cloudant.
   - Replace wluser with the name of the user who has access to the MobileFirst runtime database. If this user is not allowed to create a database, you must create a database manually, with the names that are documented in “Creating the Cloudant database” on page 11-9.
   - Replace password with the password associated with this user.

   Note: Each project using Cloudant as a database needs its own database. By default, the database includes the project name. But if several teams use the same Cloudant database with projects that have same names, add JNDI property **mfp.db.cloudant.dbNamePrefix** to distinguish the projects as indicated below, and replace your_prefix with a value of your choosing.

```
<Environment name="mfp.db.cloudant.dbNamePrefix" value="your_prefix" type="java.lang.String" override="false"/>
```

   Important: The Cloudant database name must start with a lowercase letter, and can contain only lowercase characters (a-z), digits (0-9), and any of the following characters: _, $, -.

   The value that you specify for the variable *mfp.db.cloudant.dbNamePrefix* must comply with this rule.

6. If you installed MobileFirst Operational Analytics, and you want it to collect data from the MobileFirst runtime, add the following JNDI properties:

```
<Context docBase ="worklight" path ="/worklight">
 ...
  <Environment name ="wl.analytics.url" value ="http://analytics.host:port/analytics-service/data" type ="java.lang.String" over
  <Environment name ="wl.analytics.console.url" value ="http://analytics.host:port/analytics/console" type ="java.lang.String" o
  <Environment name ="wl.analytics.username" value ="wlanalyticsuser" type ="java.lang.String" override ="false"/>
  <Environment name ="wl.analytics.password" value ="wlanalyticspassword" type ="java.lang.String" override ="false"/>
 ...
</Context>
```

   - worklight is the context root of the MobileFirst Server application. If you select another value in the previous step, replace worklight with that value.
   - Replace analytics.host:port with the host name and port of the server where you installed MobileFirst Operational Analytics.
   - Replace wlanalyticsuser with the name of the user who has access to MobileFirst Operational Analytics.
   - Replace wlanalyticspassword with the password that is associated with this user.

   Note: The user name and password that you provide in replacement of wlanalyticsuser and wlanalyticspassword must match the credentials of the user that you defined in the server where MobileFirst Operational Analytics is installed.
7. Start Tomcat.

Related reference:

"JNDI environment entries for MobileFirst projects in production" on page 11-67
JNDI environment entries cover all the properties that you can set in a production environment. Set production environment properties by configuring your project WAR file with JNDI environment entries.

Related information:

WebSphere Application Server documentation about Deploying application to the Liberty Profile

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 MobileFirst 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 MobileFirst project 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.

Declaring the Cloudant SSL configuration for a project WAR file

If you deploy a project WAR file into WebSphere Application Server or WebSphere Application Server Network Deployment, you use a Cloudant database, and meet specific criteria described in this topic, you must declare the Cloudant SSL configuration.

About this task

You must perform this procedure if you meet all of the following criteria:

- You are deploying a project WAR file into WebSphere Application Server or WebSphere Application Server Network Deployment.
- You use a Cloudant database for this web application.
- The Cloudant database server URL uses the https protocol.
- You created a separate SSL configuration for Cloudant, as described in "Configuring WebSphere Application Server for use with Cloudant" on page 6-57, but did not create a dynamic outbound configuration for it.

Procedure

1. Log in to the WebSphere Application Server console.
2. Select **Applications** > **Application Types** > **WebSphere enterprise applications**.
3. Click the web application that corresponds to the project WAR file.
4. In the **Web Module Properties** section, select **Environment entries for Web modules**.
5. Set the value of the property `mfp.db.cloudant.ssl.configuration` to the name of the SSL configuration that you created.
6. Click **OK**.
7. Click **Save**.

Configuring multiple MobileFirst 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 MobileFirst Operations Console, or with separate MobileFirst environments. An environment consists of the following web applications:

- The Administration Services web application,
- Optionally, the MobileFirst Operations Console web application,
- Some MobileFirst runtime environments or project web applications, each with the appropriate database schemas.

According to your installation method, you specify the environment as follows:

- If you install with Ant tasks, specify it through the attribute environmentId in the <installworklightadmin> and <configureapplicationserver> invocations.
- If you install manually, specify it through the JNDI property ibm.worklight.admin.environmentid of the Administration Services application, and of the project WAR file.

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 MobileFirst runtime environment must use a different runtime database or schema, and its own reports database or schema.
- Each configuration for a MobileFirst 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 MobileFirst 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 of the Ant tasks is not used in this case.
- If the application server is WebSphere Application Server full profile or WebSphere Application Server Network Deployment, each MobileFirst project must use either a different environmentId attribute or, when installed with Ant tasks and with the same environmentId, 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 MobileFirst project must use a different contextroot attribute. In addition, the versions of the JDBC drivers must be suitable for all declared data sources of the particular database type.

Configuration of MobileFirst applications on the server

You can configure each MobileFirst application by specifying a set of configuration parameters on the server side.

MobileFirst 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 MobileFirst 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. 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 MobileFirst project is built by MobileFirst Platform Command Line Interface for iOS, 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 "JNDI environment entries for MobileFirst projects in production" on page 11-67 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 11-63 to learn how to create such configuration properties.

Configuring the IBM MobileFirst Platform Server location

You can configure the MobileFirst 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 MobileFirst Server location, set the values of the following properties:

Table 11-5. MobileFirst Server location properties

Property name	Description
`publicWorkLightHostname`	The IP address or host name of the computer running IBM MobileFirst Platform Foundation for iOS. If the MobileFirst Server is behind a reverse proxy, the value is the IP address or host name of the reverse proxy. This property must be identical for nodes within the same cluster. Default: IP address of current server.

Table 11-5. MobileFirst Server location properties (continued)

Property name	Description
publicWorkLightPort	The port for accessing the MobileFirst Server. If the MobileFirst 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 MobileFirst Server. The valid values are `HTTP` and `HTTPS`. If the MobileFirst 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 “JNDI environment entries for MobileFirst projects in production” on page 11-67.

For information about how to specify configuration properties, see “Configuration of MobileFirst applications on the server” on page 11-50.

Runtime database setup for development mode

IBM MobileFirst Platform Foundation for iOS uses defaults to access the runtime database, which kind is `WRKLGHT` by default. 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. You can also use a set of properties to configure Cloudant as the runtime database.

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 11-20.

Property keys and values for JDBC-based properties

Property Key	Property Value
wl.db.url	JDBC path to the runtime database.
wl.db.username	Runtime database user name. Default: `Worklight`
wl.db.password	Runtime 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 the reports database Default: refers to runtime database. **Note:** Deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead.
wl.reports.db.username(*)	Reports database user name. Default: refers to `Worklight` database. **Note:** Deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead.
wl.reports.db.password(*)	Reports database password Default: refers to runtime database. **Note:** Deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead.

Note: (*) By default all report mechanisms in MobileFirst Server use a single reports database. The reports database is set to be the same as the runtime database. For more information about how this default setting can be changed, see "Using raw data reports" on page 13-98.

To configure the MobileFirst project to use Cloudant as the runtime database configure your Cloudant credentials (user name and password). Once these credentials are configured, the server uses Cloudant as the runtime database and not the default JPA.

In addition, you can configure the following properties:

Property keys and values for Cloudant configuration

Property Key	Property Value
mfp.db.cloudant.username	The user name of the Cloudant account.
mfp.db.cloudant.password	The password of the Cloudant account.

Property keys and values for Cloudant configuration

Property Key	Property Value
mfp.db.cloudant.url	Optional. The URL of the Cloudant account.
mfp.db.cloudant.dbNamePrefix	Optional. The prefix that is added to the database name of the MobileFirst runtime. For example, if the value department is assigned to this attribute and the context root of your MobileFirst runtime is **/mfp**, the database name is **department_mfp_runtime_db**. **Important:** This prefix must start with a lowercase letter, and can contain only lowercase characters (a-z), digits (0-9), and any of the following characters: _, $ and -.
mfp.db.cloudant.connectTimeout	Optional. The timeout to establish a connection, in milliseconds. A value of zero means an infinite timeout.
mfp.db.cloudant.socketTimeout	Optional. The timeout to wait for a response, in milliseconds. A value of zero means an infinite timeout.
mfp.db.cloudant.maxConnections	Optional. The maximum number of connections to the database.
mfp.db.cloudant.proxyHost	Optional. The host name to connect through the proxy.
mfp.db.cloudant.proxyPort	Optional. The proxy port.

Configuring session dependency

You can control session dependency by configuring the relevant properties.

The properties described in this page became available starting from IBM MobileFirst Platform Foundation for iOS V7.1.0.

Table 11-6. MobileFirst Server session dependency properties.

Property name	Description
mfp.session.independent	Turns session dependency in MobileFirst Server on or off. Options are: • true: session independent • false: session dependent Default: true

Table 11-6. MobileFirst Server session dependency properties (continued).

Property name	Description
mfp.attrStore.type	Specifies the attribute store type. Options are: • database: the attribute store is kept on the IBM MobileFirst Platform Foundation for iOS runtime database: either a relational or Cloudant database • extremescale: the attribute store is kept on eXtreme Scale • httpsession: the attribute store is kept on the HTTP session. This value is supported only if **mfp.session.independent** has been set to false, otherwise, the server will not start. • datacache: the attribute store is cached in the Data Cache service on Bluemix. Default: database

For information about configuring attribute store cleanup, see "Configuring the attribute store cleanup task."

For more information about session dependency, see "Session-independent mode" on page 8-108.

Configuring the attribute store cleanup task:

You can control the way IBM MobileFirst Platform Server cleans up stale data in the attribute store.

The properties described in this page became available starting from IBM MobileFirst Platform Foundation for iOS V7.1.0.

When the attribute store is configured to use the MobileFirst database, a cleanup task is run periodically to delete stale entries. Stale entries are attributes that have expired and have not been accessed for a period (the default is 30 days). The task is also applied when the attribute store is over eXtremeScale and the write-behind database is the MobileFirst database.

Attributes always have an expiration date (this is the expiration date of the realm with which the attributes are associated). An expired attribute is normally deleted the next time it is accessed. However, if the client application is no longer in use (for example, because it has been uninstalled), expired attributes that are associated with the client application are deleted by the cleanup task. Use the following properties to configure the cleanup:

Table 11-7. MobileFirst Server attribute store cleanup properties.

Property name	Description
mfp.attrStore.db.cleanupFrequency.minutes	The attribute store database cleanup task interval (in minutes). The default is 60 minutes.
mfp.attrStore.db.stalePeriod.days	The interval (in days) after which an expired attribute is considered stale, and may be deleted by the attribute store database cleanup task. The default is 30 days.

An example of stale data cleanup: if the **mfp.attrStore.db.cleanupFrequency.minutes** property has been set to 10 and the **mfp.attrStore.db.stalePeriod.days** property to 30, then every 10 minutes, the cleanup task runs and deletes data that expired more than 30 days ago.

For more information about session dependency, see "Session-independent mode" on page 8-108.

Configuring extended app authenticity checking

Checking extended application authenticity is an additional method of verifying that the MobileFirst application on the device has not been tampered with.

Enabling extended authenticity checking

Note: Extended application-authenticity validation is currently not supported for iOS apps that are installed from the Apple app store. See the Extended application authenticity for iOS known limitation.

To enable extended authenticity checking, you must deploy a modified `.wlapp` file, instead of the original `.wlapp` file that is generated by the build process. To obtain the modified `.wlapp` file, use one of the following two facilities:

- The **`enable-extended-authenticity`** command of the **`wladm`** Ant task, as described in "Commands for apps" on page 12-19.
- The **`enable extended-authenticity`** command of the **`wladm`** program, as described in "Commands for apps" on page 12-45.

You need to specify the following files as inputs for those commands:

- The `.wlapp` file, from the `bin` folder of your MobileFirst project.
- The mobile application file for the device, that is, the corresponding `.ipa` file. If your application must support both 32-bit and 64-bit execution, provide a single `.ipa` file that includes both 32-bit and 64-bit code.

The resulting, modified `.wlapp` file is the file that you deploy to the MobileFirst Server.

Note:

- The extended application authenticity mechanism demands that there is a dedicated version of the `.wlapp` file per each version of the native app. Before you distribute a new version of the mobile application file (`.ipa`), you must perform the following actions:
  1. Update the application version number in the `application-descriptor.xml` file.
  2. Rebuild to create a new `.wlapp` file.
  3. Run one of the commands described previously to enable authenticity checking on the production-ready `.ipa` file and the new `.wlapp` file.
  4. Deploy the `.wlapp` file produced in step 3 to the MobileFirst Server.

Disabling extended authenticity checking

To revert from extended authenticity checking to basic authenticity checking, deploy the original `.wlapp` file to the MobileFirst Server, for each of the wanted environments.

Enabling extended authenticity checking for apps that undergo app thinning

App thinning was introduced by Apple in iOS 9 for apps that are compiled with Xcode 7. App thinning reduces the size of files that are downloaded from the App Store. The feature might affect the extended authenticity features of IBM MobileFirst Platform Foundation for iOS apps because the binary file in the App Store might differ from the one that is downloaded to the client device.

Starting with the latest interim fix of IBM MobileFirst Platform Foundation for iOS V7.1.0, the new `ipa` file that you upload to the App Store can take advantage of app thinning, without influencing extended authenticity. To enable this capability, you must perform the following actions:

1. Configure the original `.wlapp` file by using the `enable-extended-authenticity` command of the `wladm` program, as described earlier in this page.
2. Deploy the `.wlapp` file to the server for testing.

Note: App thinning does not take place in apps that were developed using earlier versions of iOS and Xcode.

Push notification settings

When working with push notifications, you must use the correct proxy settings. 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`).

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`	APNS proxy user name, if the proxy requires authentication. An empty user name means no authentication.
`push.apns.proxy.password`	APNS 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.

SMS proxy settings	Value
push.sms.proxy.password	Proxy password, if the proxy requires authentication.

Analytics

Analytics properties files contain the parameters for how IBM MobileFirst Platform Foundation for iOS creates activity logs and sends them to a server for analysis.

You can modify how the MobileFirst Server forwards analytics data to the IBM MobileFirst Platform Operational Analytics by editing the following properties files.

Table 11-8. IBM MobileFirst Platform Operational Analytics properties.

Property Name	Default Value	Description
`wl.analytics.console.url`	None.	Optional. The URL that is exposed by the IBM MobileFirst Platform Operational Analytics that links to the MobileFirst Analytics Console. Set this property if you want to access the MobileFirst Analytics Console from the MobileFirst Operations Console. Example: `http://<hostname>:<port>/analytics/console`
`wl.analytics.logs.forward`	`true`	When the property is set to `true`, server logs that are recorded on the MobileFirst Server are captured and forwarded to the IBM MobileFirst Platform Operational Analytics.
`wl.analytics.password`	None.	Required. The password that is used if the data entry point for the IBM MobileFirst Platform Operational Analytics is protected with basic authentication.
`wl.analytics.url`	None.	Required. The URL that is exposed by the IBM MobileFirst Platform Operational Analytics that receives incoming analytics data. Example: `http://<hostname>:<port>/analytics-service/d`
`wl.analytics.username`	None.	Required. The user name that is used if the data entry point for the IBM MobileFirst Platform Operational Analytics is protected with basic authentication.

WebSphere Application Server SSL configuration and 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 MobileFirst adapters and back-end servers by using self-signed certificates” on page 6-178. 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 as 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.

SSL certificate keystore setup

Mobile applications often connect to multiple back-end systems. Some back-end systems require access through an HTTP adapter, and each back-end system can require a different SSL certificate for secure communication using HTTPS. These SSL certificates are stored in a keystore that is configured to the IBM MobileFirst Platform Server by using property keys.

IBM MobileFirst Platform Foundation for iOS 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 11-9. JNDI environment entries for SSL certificate keystore

Property name	Description
ssl.keystore.path	Path to the keystore relative to the server folder in the MobileFirst 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.alias	WebSphere SSL configuration alias used by the HTTP adapters
ssl.websphere.config	Set this property to true to have HTTP adapters use WebSphere SSL configuration. Default: false.

For descriptions of other MobileFirst configuration properties, see “JNDI environment entries for MobileFirst projects in production” on page 11-67.

For information about how to specify MobileFirst configuration properties, see "Configuration of MobileFirst applications on the server" on page 11-50.

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 "Structure of the adapter XML file" on page 8-38.

If you use SSL with mutual authentication between the MobileFirst 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 "HTTP adapter connectionPolicy element" on page 8-43.

  Note: The password that is specified in **ssl.keystore.password** is not the same password that is specified in <sslCertificatePassword>. **ssl.keystore.password** is used to access the keystore itself. <sslCertificatePassword> is used to access the correct SSL certificate within the keystore.

Miscellaneous settings

Configure the **serverSessionTimeout**, **bitly.username**, **bitly.apikey**, **compress.response.threshold**, and **adapters.saxparser.doctype.validation** parameters in the worklight.properties file.

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 MobileFirst session is invalidated. Default is 10 minutes. **Note:** Session timeout configuration is not used in session-independent mode.
bitly.username	User name for accessing the bit.ly API for creating a shortened URL for mobile web apps through MobileFirst Operations 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.

Property keys and values for the **serverSessionTimeout**, **bitly.username**, **bitly.apikey**, **compress.response.threshold**, and **adapters.saxparser.doctype.validation** parameters.

Property Key	Property Value
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

When you configure MobileFirst applications on the server, you must encrypt the properties that are too sensitive to be written in clear text.

You can encrypt properties in two ways:

- Within the properties file: See "Encryption within the properties file." This option is the only one for Tomcat.
- By using the application server encoding tools: PropFilePasswordEncoder for WebSphere Application Server and SecurityUtility for Liberty profile. For WebSphere Application Server and Liberty profile. See "Encoding the JNDI properties" on page 11-62.

Encryption within the properties file

The encryption facility that comes with IBM MobileFirst Platform Foundation for iOS uses the 128-bit symmetric-key algorithm that is defined by the AES specification.

Storing properties in open or encrypted format

You can keep the properties that are contained in the worklight.properties file either in open or in encrypted form.

An encrypted property is determined by a suffix .enc appended to its name. For example:

```
console.password.enc=TYakEHRba3rIU7pNjxtDxoAdqijKIEt7cy4mCr0iaEj0rY08ODK00yqR
```

The MobileFirst configuration is accessed for a property. If the property is not found, but the same encrypted property (with the .enc suffix) is defined, MobileFirst automatically decrypts the value, and returns it to the caller.

Storing the master key

All 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 that runsMobileFirst Server. 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 MobileFirst properties by using the 128-bit symmetric-key algorithm that is defined by the AES specification.

- On Windows systems, use the `encrypt.bat` utility under *product_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 you run the `encrypt.sh` tool, the `secret.properties` file 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 JNDI properties in WebSphere Application Server is to use the password encoding tools that are available with both application servers.

- For WebSphere Application Server: the `PropFilePasswordEncoder` tool
- For the Liberty profile: the `SecurityUtility` command. For the encoding type, only `xor` and `aes` with the default key are supported.

You can use the encoded value as the value of the JNDI properties.

For more information about how to encode properties with the application server tools, see the WebSphere Application Server documentation.

Obsolete properties

Some properties are no longer required.

Table 11-10. 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.iphone**, **environment.embedded**
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**

Table 11-10. Categories and list of obsolete properties (continued)

Category	Properties
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 MobileFirst application so that an IT administrator can change the value in production. For example, a MobileFirst 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 MobileFirst 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 "JNDI environment entries for MobileFirst projects in production" on page 11-67) 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 8-271. For **loginModule** parameters, see "Configuring login modules" on page 8-263.

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 certificate from key store  -->

    <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 in the project's lib folder -->
  <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 "HTTP adapter `connectionPolicy` element" on page 8-43.

Configuring a MobileFirst project in production by using JNDI environment entries

When you deploy a MobileFirst project to MobileFirst Server, you can configure the project WAR file with JNDI environment entries to set product environment properties.

About this task

JNDI environment entries cover all the properties that you can set in a production environment. For details about the JNDI entries, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

Procedure

Set the JNDI environment entries in one of the following ways:

- Edit the configuration XML file for the deployer Ant tasks.

  When you deploy and configure the project with the Ant task, you can set values for MobileFirst configuration properties inside the `<configureapplicationserver>` tag. For more information, see "Deploying a project WAR file and configuring the application server with Ant tasks" on page 11-16 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"/>
  ```

- Configure the server environment entries. The steps to configure the server environment entries depends on which application server you use.
  - 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 server environment.

– WebSphere Application Server 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 MobileFirst application. If multiple MobileFirst 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.

Preconfiguring JNDI properties:

As an alternative to setting JNDI environment entries when you deploy a MobileFirst project to MobileFirst Server, you can configure all JNDI properties before you deploy by using a property file. Holding JNDI properties in a property file makes it easier to transfer the entire configuration from one web application server to another. For example, you can configure a test web server; when the configuration is stable, you can easily transfer the configuration to the production web server by copying the property file to the production server. For more information, see "Predefining MobileFirst Server configuration for several deployment environments" on page 6-287.

In some cases, when you do not want to or cannot redeploy the application, you can set values for MobileFirst 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 you upgrade or redeploy), the application server might forget the configuration and the administrator must reconfigure it.

Tip: If you choose to configure manually, be aware that if the application is upgraded or redeployed, your settings might be overwritten and then the administrator would then have to reconfigure the properties.

Related reference:

"Configuration of MobileFirst applications on the server" on page 11-50
You can configure each MobileFirst application by specifying a set of configuration parameters on the server side.

"JNDI environment entries for MobileFirst projects in production"
JNDI environment entries cover all the properties that you can set in a production environment. Set production environment properties by configuring your project WAR file with JNDI environment entries.

JNDI environment entries for MobileFirst projects in production:

JNDI environment entries cover all the properties that you can set in a production environment. Set production environment properties by configuring your project WAR file with JNDI environment entries.

JNDI environment entries cover all the properties that you can set in a production environment. You set the JNDI environment entries in one of two ways:

- Editing the configuration XML file for the deployer Ant tasks.
- Configuring the server environment entries. On WebSphere Application Server full profile and WebSphere Application Server Network Deployment, use the administration console. On WebSphere Application Server Liberty profile or Apache Tomcat, you edit the `server.xml` file.

For procedure details, see "Configuring a MobileFirst project in production by using JNDI environment entries" on page 11-65.

Many of the MobileFirst 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 MobileFirst Server public URL (that is, **`publicWorkLightHostname`**, **`publicWorkLightPort`**, and **`publicWorkLightProtocol`**) might be different when the MobileFirst project is deployed to a staging server or to a production server. You can configure the project WAR file through JNDI environment entries.

Some of the properties are relevant only in a development environment and are not available as JNDI entries.

You can encrypt the JNDI properties that are listed in the table that follows in two ways:

- You can define the property with the `.enc` suffix in the `worklight.properties` file that is packaged in the WAR file of the MobileFirst 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.

For more information, see "Storing properties in encrypted format" on page 11-61.

The following table lists the MobileFirst properties that are always available as JNDI entries:

Table 11-11. MobileFirst 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 you set this property to `false`, the adapter does not validate the response. This might be useful in cases when the time required to validate could be expected to exceed the allowed timeout value. Default: `true`

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
cluster.data.synchronization.taskFrequencyInSeconds	Interval for the synchronization of event sources data within clusters. Default: 2.
deployables.cleanup.taskFrequencyInSeconds	Interval at which deployable folder are cleaned up (in seconds). Default: 86400.
ibm.worklight.admin.environmentid	Optional. Environment identifier for the registration of the MBeans. Use this identifier when different instances of the MobileFirst 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.jmx.connector	Mandatory. JMX connector type, by default RMI or SOAP. WebSphere Application Server 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.jmx.pwd	Optional. WebSphere Application Server Farm: the user password of the SOAP connection.
ibm.worklight.admin.jmx.user	Optional. WebSphere Application Server Farm: the user name of the SOAP connection.
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.serverid	Optional. Server identifier. Must be different for each server in the farm. Server farms only.
ibm.worklight.jndi.configuration	Optional. If the JNDI configuration is injected into the WAR files or provided as a shared library, the value of this property is the name of the JNDI configuration. You can also specify this value as a system property. See "Predefining MobileFirst Server configuration for several deployment environments" on page 6-287.
ibm.worklight.jndi.file	Optional. If the JNDI configuration is stored as an external file, the value of this property is the path of a file that describes the JNDI configuration. You can also specify this value as a system property. See "Predefining MobileFirst Server configuration for several deployment environments" on page 6-287.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
ibm.worklight.topology.clustermode	In addition to the server type, you must specify the server topology. Valid values: • Standalone • Cluster • Farm The default value is Standalone.
ibm.worklight.topology.platform	Server type. Valid values: • Liberty • WAS • Tomcat If the default value is not set, the application tries to guess the server type.
publicWorkLightHostname	The IP address or host name of the computer that is running IBM MobileFirst Platform Foundation for iOS. If the MobileFirst 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 MobileFirst Server. If the MobileFirst 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 MobileFirst Server. The valid values are HTTP and HTTPS. If the MobileFirst 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.
push.apns.proxy.enabled	Indicates whether APNS must be accessed through a proxy. Default: false.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
push.apns.proxy.host	The hostname of the proxy server to be used to connect to APNS.
push.apns.proxy.port	The port number on the proxy server to be used to connect to APNS.
push.apns.proxy.user	The user name for authenticating with the APNS proxy server if the proxy requires authentication. An empty user name means no authentication.
push.apns.proxy.password	The password for authenticating with the APNS proxy if the proxy requires authentication.
push.clearUserOnLogout	Clear the user name from push device registration on logout. Default: `false`
push.sms.proxy.enabled	Indicates whether push SMS proxy is enabled. Default: `false`.
push.sms.proxy.host	The hostname of the proxy server to be used to connect to the SMS Gateway.
push.sms.proxy.password	The password for authenticating with the proxy server if authentication is enabled on the proxy
push.sms.proxy.port	The port number on the proxy server to be used to connect to the SMS Gateway
push.sms.proxy.protocol	The protocol to be used (HTTP or HTTPS) to connect to the proxy server
push.sms.proxy.user	The user name for authenticating with the proxy server if authentication is enabled on the proxy
serverSessionTimeout	Idle session timeout in minutes. Default: `10`. Used in session-dependent mode only.
ssl.keystore.password	SSL certificate keystore password.
ssl.keystore.path	SSL certificate keystore location. Default: `conf/mfp-default.keystore`.
ssl.keystore.type	SSL certificate keystore type. Valid keystore types: `jks` or `PKCS12`. Default: `jks`.
ssl.websphere.alias	The WebSphere SSL configuration alias used by the HTTP adapters
ssl.websphere.config	Set this property to `true` to have HTTP adapters use WebSphere SSL configuration. Default: `false`.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
sso.cleanup.taskFrequencyInSeconds	Interval (seconds) for a cleanup task that cleans the database of orphaned and expired single-sign-on login contexts. Default: 5
trusted.signer.certificate.paths	A space-separated list of trusted signer certificates for authenticating non-mobile (confidential) clients. The paths may be relative to the server folder in the MobileFirst Project (for example: `conf/rootCA.crt`), or absolute paths.
wl.analytics.console.url	Optional. The URL that is exposed by IBM MobileFirst Platform Operational Analytics that links to the Analytics console. Set this property if you want to access the Analytics console from the MobileFirst Operations Console. Example: `http://<hostname>:<port>/<context-root>/console`
wl.analytics.logs.forward	A 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 supported only on MobileFirst production servers.
wl.analytics.password	The password that is used if the data entry point for the IBM MobileFirst Platform Operational Analytics is protected with basic authentication.
wl.analytics.url	The URL that is exposed by the IBM MobileFirst Platform Operational Analytics 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 MobileFirst Platform Operational Analytics is protected with basic authentication.
wl.ca.key.alias	The alias of the entry where the private key and certificate are stored in the keystore. Default: `mfp-default-cert`.
wl.ca.key.alias.password	The password that protects the keystore entry where the private key and certificate are stored. The alias name for this entry is defined in the **wl.ca.key.alias** JNDI property.
wl.ca.keystore.password	The password that protects the keystore. The path to the keystore is defined in the **wl.ca.keystore.path** JNDI property.
wl.ca.keystore.path	The path to the keystore relative to the server folder in the MobileFirst project. Default: `conf/mfp-default.keystore`.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
wl.ca.keystore.type	Type of keystore file. Valid values are jks or pkcs12. Default: jks.
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.
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 MobileFirst 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 MobileFirst Server. If the Access Management features are enabled, each time a device attempts to connect to the server, it is checked against the back end for its access rights.
wl.device.tracking.enabled	A value that is used to enable or disable device tracking in IBM MobileFirst Platform Foundation for iOS. For performance reasons, you can disable this flag when IBM MobileFirst Platform Foundation for iOS runs only Business-to-Consumer (B2C) apps. When device tracking is disabled, the license reports are also disabled and no license metrics are generated.
mfp.adapter.invocation.url	The URL to be used for invoking adapter procedures from inside Java adapters, or JavaScript adapters that are invoked using the /rest endpoint. If this property is not set, the URL of the currently executing request will be used (this is the default behavior). This value should contain the full URL, including the context root.
mfp.attrStore.db.cleanupFrequency.minutes	The attribute store database cleanup task interval (in minutes). The default is 60 minutes.
mfp.attrStore.db.stalePeriod.days	The interval (in days) after which an expired attribute is considered stale, and may be deleted by the attribute store database cleanup task. The default is 30 days. For more information about the store database cleanup task, see “Configuring the attribute store cleanup task” on page 11-55.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
mfp.attrStore.type	Specifies the cache or persistent store to save the authentication context that is kept in the attribute store of MobileFirst Server. Options: • database: This is the default option. Uses the same persistent store that is configured for the runtime database (either a relational or Cloudant database) • extremescale: uses eXtreme Scale as a data cache • httpsession: uses the HTTP session to store authentication context, as it was used in MobileFirst Server versions Before V7.1.0. Use this option only if **mfp.session.independent** has been set to false, otherwise, the server will not start. • datacache: relevant only when working in Bluemix Default: database
mfp.attrStore.xs.endpoint	A comma-separated list of host-port pairs of the WebSphere eXtreme Scale catalog servers, in the form: "host:port".
mfp.attrStore.xs.gridname	The eXtreme Scale grid that is used for caching the attribute store.
mfp.attrStore.xs.mapname	The eXtreme Scale backing map that is used for caching the attribute store.
mfp.attrStore.xs.password	The password for connecting to the eXtreme Scale server. Leave empty if authentication is not required.
mfp.attrStore.xs.username	The user name for connecting to the eXtreme Scale server. Leave empty if authentication is not required.
mfp.db.cloudant.username	The user name of the Cloudant account. When this property is defined along with a Cloudant password, the runtime database is Cloudant. Otherwise, the runtime uses the relational database that is configured.
mfp.db.cloudant.password	The password to the Cloudant account. When this property is defined along with a Cloudant user name, the runtime database is Cloudant. Otherwise, the runtime uses the relational database that is configured.
mfp.db.cloudant.url	The URL of the Cloudant account that is used only if Cloudant is active, and has a defined username and password. When this property is defined, the Cloudant database is directed to this URL. Otherwise, it is redirected to the default URL: https://cloudant.com
mfp.db.cloudant.ssl.authentication	A Boolean value (true or false) that specifies whether the SSL certificate chain validation and host name verification are enabled for HTTPS connections to the Cloudant database. Default: true. **Important:** Setting this property to false creates security risks.

Table 11-11. MobileFirst properties available as JNDI entries (continued).

Property name	Description
mfp.db.cloudant.ssl.configuration	Property that applies to WebSphere Application Server Full Profile only. For HTTPS connections to the Cloudant database, it specifies the name of an SSL configuration in the WebSphere Application Server configuration, to use when no configuration is specified for the host and port.
mfp.db.cloudant.dbNamePrefix	The Cloudant database prefix. When this property is configured, the prefix is added to the default database name, followed by an underscore: "*MobileFirst_runtime_context-root*_runtime_db". The prefix string must contain lowercase letters only.
mfp.db.cloudant.connectTimeout	A timeout for establishing a network connection to Cloudant, in milliseconds. The zero (0) value means an infinite timeout. A negative value means the default (no override). If this property is not configured, a default value is used, which is described in the documentation for the Cloudant Java Client.
mfp.db.cloudant.socketTimeout	A timeout for detecting the loss of a network connection to Cloudant, in milliseconds. The zero (0) value means an infinite timeout. A negative value means the default (no override). If this property is not configured, a default value is used, which is described in the documentation for the Cloudant Java Client.
mfp.db.cloudant.maxConnections	The maximum number of connections of the Cloudant connector. If this property is not configured, a default value is used, which is described in the documentation for the Cloudant Java Client.
mfp.db.cloudant.proxyHost	The proxy host of the Cloudant connector. If this property is not configured, a default value is used, which is described in the documentation for the Cloudant Java Client.
mfp.db.cloudant.proxyPort	The proxy port of the Cloudant connector. If this property is not configured, a default value is used, which is described in the documentation for the Cloudant Java Client.
mfp.session.independent	Turns session dependency in MobileFirst Server on or off. Options: • `true`: session independent • `false`: session dependent Default: `true`

Custom user properties that are defined in the `worklight.properties` file are also exposed.

The **wl.db.*** and **wl.reports.db.*** properties are not available as JNDI environment entries because they are intended for use only during the development phase.

Related tasks:

"Configuring a MobileFirst project in production by using JNDI environment entries" on page 11-65
When you deploy a MobileFirst project to MobileFirst Server, you can configure the project WAR file with JNDI environment entries to set product environment properties.

Related reference:

List of JNDI properties for MobileFirst server administration
When you configure MobileFirst Server Administration Services and MobileFirst Operations Console for your application server, you set optional or mandatory JNDI properties, in particular for Java Management Extensions (JMX).

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 MobileFirst Platform Foundation for iOS routes the SMS notification messages through the SMS gateway.

To send SMS notifications from IBM MobileFirst Platform Foundation for iOS, 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 MobileFirst Server must be restarted when any changes are made in the `SMSConfig.xml` file.

Table 11-12. `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 11-13 on page 11-76.

Table 11-12. `SMSConfig.xml` elements and subelements (continued)

Element	Element Value
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 11-13. `<gateway>` element attributes

Attribute	Attribute Value
hostname	Mandatory. The host name of the configured SMS gateway.
id	Mandatory. A unique ID that identifies the SMS gateway. Application developers specify the ID in the application descriptor file, `application-descriptor.xml`, when they develop an application.
port	Optional. The port number of the SMS gateway. The default value is 80.
programName	Optional. The name of the program that the SMS gateway expects. For example, if the SMS gateway expects the following URI: `http://<hostname>:port/sendsms` then **programName**="sendsms"
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/XMLSchema-instance">

  <gateway hostname="myhost" id="kannelgw" port="13011" programName="cgi-bin/sendsms" toParamName="to" textParamName="text">
   <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 MobileFirst Server and the IBM MobileFirst Platform Command Line Interface for iOS. You can use them to build and deploy your applications, adapters, and projects.

IBM MobileFirst Platform Foundation for iOS provides a set of Ant tasks that help you build and deploy adapters and applications to your MobileFirst Server. A typical use of these Ant tasks is to integrate them with a central build service that is called manually or periodically on a central build server.

Prerequisites

Before you can run Ant tasks, make sure that Apache Ant is installed. The minimum supported version of Ant is listed in "System requirements" on page 2-9.

Apache Ant 1.8.4 is included in MobileFirst Server. In the *product_install_dir*/shortcuts/ directory, the following scripts are provided.
- For UNIX / Linux: ant
- For Windows: ant.bat

These scripts are ready to run, which means that they do not require specific environment variables. If the JAVA_HOME environment variable is set, the scripts accept it.

Building applications and adapters

If you use Ant tasks, you can build both applications and adapters.

You can use the following examples of Ant XML files to build applications and adapters.

Note: Since V6.2.0, the worklight-ant-builder.jar file is included in the IBM MobileFirst Platform Command Line Interface for iOS as well as the MobileFirst Server, whereas in earlier versions, it was included only in Worklight Server. By default, worklight-ant-builder.jar is installed in the following location: *cli_install_dir*/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.

In Worklight Server V6.2.0 and MobileFirst Server, the default location of worklight-ant-builder.jar is *mfp_server_install_dir*/WorklightServer/worklight-ant-builder.jar. This file has the same version as the server.

Building a native API application

The Ant task for building a native API 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="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 outputFolder 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 version="1.0" encoding="UTF-8"?>
<project basedir="." default="target-name">
  <taskdef resource="com/worklight/ant/defaults.properties">
    <classpath>
      <pathelement location="cli_install_dir/public/worklight-ant-builder.jar"/>
    </classpath>
  </taskdef>

  <path id="base.path">
    <pathelement path="/Users/miguel/badjars"/>
  </path>

  <target name="target-name">
    <adapter-builder
      folder="adapter-source-files-folder"
      destinationfolder="destination-folder"
      classpathref="base.path"/>
  </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.
- The classpathref attribute specifies a custom class path for the adapter builder. If this property is not specified, then the class path of the running Ant task is used to compile the code.

The classpathref attribute is required to be set for RESTful Java adapters. It is not mandatory for JavaScript adapters. You must add the Worklight-jee-library.jar to the class path for Java adapters. In the case of RESTful Java adapters, the task compiles the code in the *adapterRootFolder*/src directory and copies the compiled classes to the adapterRootFolder/bin directory. Any source files that are not Java are also copied. Error messages are printed in the console.

This Ant task requires that the JAVA_HOME environment variable is set to point to a JDK and not a JRE.

If you must build more than one adapter file, add an <adapter-builder> element for each adapter.

Example:

Building a JavaScript adapter

```
<adapter-builder
  folder="adapterRootFolder"
  destinationfolder="destination-folder"/>
```

Building a Java adapter without libs

```
<path id="my.path">
  <pathelement path="/product_install_dir/WorklightServer/worklight-jee-library.jar"/>
</path>

<adapter-builder
  folder="adapterRootFolder"
  destinationfolder="destination-folder"
  classpathref="my.path"/>
```

Building a Java adapter with the server/lib and the adapter/lib folders in the class path

```
<path id="my.path">
  <pathelement path="/product_install_dir/WorklightServer/worklight-jee-library.jar"/>
  <fileset dir="adapterRootFolder/lib">
    <include name="*.jar" />
  </fileset>
  <fileset dir="projectRootFolder/server/lib">
    <include name="*.jar" />
  </fileset>
</path>

<adapter-builder
  folder="adapterRootFolder"
  destinationfolder="destination-folder"
  classpathref="my.path"/>
```

Building the Starter project

To use the Ant script starter example, complete the following steps.

1. Download the Starter Application sample.
2. Add the build.xml file to the root folder of your project.
3. Complete the following properties inside the build.xml file with the correct values. These values depend on your computer and the location where IBM MobileFirst Platform Foundation for iOS is installed.

```
<!-- The admin console URL -->
<property name="mfp.admin.url" value="http://localhost:10080/worklightadmin"/>
<!-- The MobileFirst server host to be written in the app configuration file (wlconfig.properties or worklight.plist) -->
<property name="mfp.server.url" value="http://localhost:10080/worklight"/>
<!-- The MobileFirst builder jar - This jar contains all the builder ant tasks -->
<property name="mfp.ant.builder.jar" value="product_install_dir/WorklightServer/worklight-ant-builder.jar"/>
<!-- The MobileFirst deployer jar - This jar contains all the deployer ant tasks -->
<property name="mfp.ant.deployer.jar" value="product_install_dir/WorklightServer/worklight-ant-deployer.jar"/>
<!-- The MobileFirst server Java EE library jar - This jar is required for building the server/java folder
and for building Java adapters as well -->
<property name="mfp.jee.library.jar" value="product_install_dir/WorklightServer/worklight-jee-library-production.jar"/>
```

4. Ensure that the MobileFirst Server is running and that MobileFirst WAR is installed on it.

5. Run the **ant** command. The **ant** command builds and deploys the app and the adapter of the Starter Application project. It also builds, but does not deploy, the WAR.

```
<?xml version="1.0"?>
  <project default="build-all-and-deploy" name="build-tools" basedir=".">
    <property name="mfp.runtime.name" value="worklight"/>
      <!-- The admin console URL -->
    <property name="mfp.admin.url" value="http://localhost:10080/worklightadmin"/>
      <!-- The MobileFirst server host to be written in the app configuration file (wlconfig.properties or worklight.plist) -->
    <property name="mfp.server.url" value="http://localhost:10080/worklight"/>
      <!-- The MobileFirst builder jar - This jar contains all the builder ant tasks -->
    <property name="mfp.ant.builder.jar" value="product_install_dir/WorklightServer/worklight-ant-builder.jar"/>
      <!-- The MobileFirst deployer jar - This jar contains all the deployer ant tasks -->
    <property name="mfp.ant.deployer.jar" value="product_install_dir/WorklightServer/worklight-ant-deployer.jar"/>
      <!-- The MobileFirst server Java EE library jar - This jar is required for building the server/java folder
and for building Java adapters as well -->
    <property name="mfp.jee.library.jar" value="product_install_dir/WorklightServer/worklight-jee-library-production.jar"/>
    <taskdef resource="com/worklight/ant/defaults.properties">
      <classpath>
        <pathelement location="${mfp.ant.builder.jar}"/>
      </classpath>
    </taskdef>
    <taskdef resource="com/worklight/ant/deployers/antlib.xml">
      <classpath>
        <pathelement location="${mfp.ant.deployer.jar}"/>
      </classpath>
    </taskdef>
    <target name="build-all-and-deploy">
      <antcall target="build-war"/>
      <antcall target="build-applications"/>
      <antcall target="build-adapters"/>
      <antcall target="deploy"/>
    </target>
    <target name="build-war">
      <path id="compiler-classpath">
        <pathelement location="${mfp.jee.library.jar}"/>
      </path>
      <mkdir dir="bin/classes"/>
      <javac classpath="${mfp.jee.library.jar}" destdir="bin/classes" srcdir="server/java">
        <classpath refid="compiler-classpath"/>
      </javac>
      <war-builder classesFolder="bin/classes" warfile="bin/StarterApplication.war" destinationfolder="bin" projectfolder="."/>
    </target>
  <target name="build-applications">
    <echo>Building Hybrid application "StarterApplication"...</echo>
    <app-builder environments="android,iphone,windowsphone8,desktopbrowser,ipad,mobilewebapp,windows8,windowsphoneunversal,common"
      nativeprojectprefix="StarterApplication" outputfolder="bin" worklightserverhost="${mfp.server.url}"
applicationfolder="apps/StarterApplication"/>
    <echo>Building Native iOS application "NativeiOS"...</echo>
    <native-app-builder worklightserverhost="${mfp.server.url}" outputFolder="bin" sourcefolder="apps/NativeiOS"/>
    <echo>Building Native Android application "NativeAndroid"...</echo>
    <native-app-builder worklightserverhost="${mfp.server.url}" outputFolder="bin" sourcefolder="apps/NativeAndroid"/>
    <echo>Building Native WP8 application "NativeWP8"...</echo>
    <native-app-builder worklightserverhost="${mfp.server.url}" outputFolder="bin" sourcefolder="apps/NativeWP8"/></target>
  <target name="build-adapters">
    <antcall target="build-adapter">
      <param name="adapter.folder" value="adapters/StarterApplicationAdapter"/>
    </antcall>
  </target>
  <target name="build-adapter">
    <path id="compiler-classpath">
      <fileset dir="${adapter.folder}/lib">
        <include name="*.jar"/>
      </fileset>
      <fileset dir="server/lib">
        <include name="*.jar"/>
      </fileset>
      <pathelement location="${mfp.jee.library.jar}"/>
    </path>
    <echo>Building adapter ${adapter.name}</echo>
    <adapter-builder destinationfolder="bin" classpathref="compiler-classpath" folder="${adapter.folder}"/>
  </target>
  <target name="deploy">
    <wladm password="admin" user="admin" secure="false" url="${mfp.admin.url}">
      <deploy-adapter file="bin/StarterApplicationAdapter.adapter" runtime="${mfp.runtime.name}"/>
      <deploy-app file="bin/StarterApplication-all.wlapp" runtime="${mfp.runtime.name}"/>
    </wladm>
  </target>
</project>
```

Building applications from IBM Worklight V6.0.0 and deploying them to MobileFirst Server

Users who want to build applications and adapters from IBM Worklight V6.0.0 to their current version of MobileFirst Server must use two Ant scripts, one to build their artifacts and one to deploy them.

If you want to build apps and adapters from a IBM Worklight V6.0.0 project and deploy them to a MobileFirst Server, you might think that all you need to do is add a `<taskdef>` definition as shown in the following Ant task.

Note:

- *`WL600_DIR`* is the directory where you installed IBM Worklight V6.0.0.
- *`product_install_dir`* is the directory where you installed the current version of MobileFirst Server.

```
<taskdef resource="com/worklight/ant/defaults.properties">
  <classpath>
    <pathelement location="WL600_DIR/WorklightServer/worklight-ant.jar" />
  </classpath>
</taskdef>
<taskdef resource="com/worklight/ant/deployers/antlib.xml">
  <classpath>
    <pathelement location="product_install_dir/WorklightServer/worklight-ant-deployer.jar" />
  </classpath>
</taskdef>
```

However, the JAR files `worklight-ant.jar` and `worklight-ant-deployer.jar` conflict, because they contain classes with the same name in different versions. 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 MobileFirst 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}" destinationFolder="${output.location}"/>
    <app-builder applicationFolder="${wl.project.location}/apps/${wl.application.name}" outputfolder="${output.location}"
worklightserverhost="${wl.server}" nativeprojectprefix="${project.name}"/>
  </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 MobileFirst 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">
    <wladm url="https://server-address:secure-port/worklightadmin" user="username" password="password">
      <deploy-app runtime="project-name" file="${output.location}/${wl.application.name}-all.wlapp"/>
      <deploy-adapter runtime="project-name" file="${output.location}/${wl.adapter.name}.adapter"/>
    </wladm>
  </target>
</project>
```

Deploying applications and adapters

You can use Ant tasks to deploy MobileFirst applications and adapters.

The following sections show examples of Ant XML files that use the **wladm** Ant task to deploy applications and adapters. You can run these Ant files locally on the MobileFirst Server host computer or remotely on a different computer. To run them remotely on a different computer, you must first copy the file *product_install_dir*/WorklightServer/worklight-ant-deployer.jar to that computer.

Deploying an application

Note: Before you use this Ant task, as a prerequisite step, you must deploy the corresponding MobileFirst project of the application. For more information, see "Deploying the project WAR file" on page 11-5.

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="product_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:

Table 11-14. Attributes of the <wladm> element

Attribute	Mandatory/ Optional	Description
url	Mandatory	The full URL of your MobileFirst Server web application for administration services
user and **password**	Mandatory	The credentials of a user in a **worklightadmin** or **worklightdeployer** role

The <deploy-app> element has the following attributes:

Table 11-15. Attributes of the <deploy-app> element

Attribute	Mandatory/ Optional	Description
runtime	Mandatory	The name of the MobileFirst runtime / project.
file	Mandatory	Contains the .wlapp file to deploy.

For more information about <wladm>, see "Administering MobileFirst applications through Ant" on page 12-12.

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: Before you use this Ant task, as a prerequisite step, you must deploy the corresponding MobileFirst project of the adapter. For more information, see "Deploying the project WAR file" on page 11-5.

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="product_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-adapter runtime="project-name" file="myAdapter.adapter"/>
    </wladm>
  </target>
</project>
```

The <wladm> element has the following attributes:

Table 11-16. Attributes of the <wladm> element

Attribute	Mandatory/ Optional	Description
url	Mandatory	The full URL of your MobileFirst Server web application for administration services
user and **password**	Mandatory	The credentials of a user in a **worklightadmin** or **worklightdeployer** role

The <deploy-adapter> element has the following attributes:

Table 11-17. Attributes of the <deploy-adapter> element

Attribute	Mandatory/ Optional	Description
runtime	Mandatory	The name of the MobileFirst runtime / project.
file	Mandatory	Contains the .adapter file to deploy.

For more information about <wladm>, see "Administering MobileFirst applications through Ant" on page 12-12.

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 MobileFirst 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 IBM MobileFirst Platform Server and their underlying integration adapters. You can create apps and adapters by building them using IBM MobileFirst Platform Command Line Interface for iOS, or with the Ant tasks provided with IBM MobileFirst Platform Foundation for iOS to build them. The result of the build action is files with extension .wlapp and .adapter respectively.

There are two ways to deploy applications and adapters to IBM MobileFirst Platform Operations Console:

- Use Ant tasks that are provided with IBM MobileFirst Platform Foundation for iOS, and described in "Ant tasks for building and deploying applications and adapters" on page 11-77 and "Deploying a project WAR file and configuring the application server with Ant tasks" on page 11-16.
- Use MobileFirst Operations Console to manually deploy apps and adapters.

You can deploy customer-specific content (apps and adapters) only after the project and MobileFirst administration WAR files are deployed and the server is started.

Note: If your browser supports multiple files upload, you can select several adapters or several applications in the file chooser dialog.

Procedure

This procedure describes how to deploy applications and adapters by using MobileFirst Operations Console.

1. To deploy one or several adapters, click **Add new app or adapter**. Then, navigate to each required adapter file and select it. A message is displayed that indicates whether the deployment action succeeded or failed.
2. Click **Adapters** to display the adapter list.
3. In the adapter list, expand the deployed adapter to display connectivity details.
4. Repeat steps 1 to 3 for each adapter that you want to deploy.

5. To deploy an application, either deploy the *app_name*-all.wlapp file, or individual *app_name-environment_name-version*.wlapp files. Click **Add new app or adapter**. Then, navigate to the WLAPP files and select them. A message is displayed that indicates whether the deployment action succeeded or failed.
6. In the application list, click the deployed application **summary** link to display details of environments and versions.
7. Repeat steps 5 and 6 for each app that you want to deploy.

Administering adapters and apps in MobileFirst Operations Console

You administer adapters and apps through MobileFirst Operations Console.

About this task

Before performing any of the other tasks in this collection of topics, open MobileFirst Operations 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 MobileFirst Operations Console through http instead of https, your MobileFirst administration user password will be compromised.

   This usage is different from the MobileFirst Development Server, 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 MobileFirst 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 MobileFirst Operations Console and you can start performing administration tasks.

If several projects are deployed on the server, you see a list of projects in MobileFirst Operations Console. Select the project to administer to navigate to the Catalog page of this project.

Deploying apps

You can select WLAPP files to deploy.

About this task

If your browser supports multiple files upload, you can select several applications in the file chooser dialog.

Procedure

To deploy an app, you deploy either the *app_name*-all.wlapp file or individual *app_name-environment_name-version*.wlapp files.

1. From the first page of MobileFirst Operations Console, click **Add new app or adapter**.
2. Select adapter files for deployment in one of the following ways
   - Select one or several WLAPP files.
   - If your browser is recent enough, drag one or several WLAPP files from the file navigator in your operating system into the first page of MobileFirst Operations Console that is open in the browser.

   A message is displayed that indicates whether the deployment action succeeded or failed.

Deleting apps

You can delete an app or a version of an environment of an app.

About this task

An application can be provided in different mobile device environments, such as a tablet version and a phone version. Note that deleting an application from MobileFirst Operations Console will remove all push subscriptions on that application as well.

Procedure

1. From the first page of MobileFirst Operations Console, click the application name to go the application details page.
2. Process the deletion:
   - To delete an app, click **Delete application**.
   - To delete a version of an environment of an app, click **Delete version**.

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 MobileFirst Operations Console.

About this task

If your browser supports multiple files upload, you can select several adapters in the file chooser dialog.

Procedure

To deploy an adapter:

1. From the first page of MobileFirst Operations Console, click **Add new app or adapter**.
2. Select adapter files for deployment in one of the following ways

- Select one or several adapter files.
- If your browser is recent enough, drag one or several adapter files from the file navigator in your operating system into the first page of MobileFirst Operations Console that is open in the browser.

A message is displayed that indicates whether the deployment action succeeded or failed.

3. Expand the deployed adapters to view the connectivity details and the list of procedures that it shows.

Modifying adapters

To modify an adapter, replace it with a new one.

Procedure

To modify an adapter:

Deploy the modified adapter file, as described in “Deploying adapters” on page 11-86.

Results

The new adapter replaces the original one.

Deleting adapters

You can delete an adapter from MobileFirst Operations Console.

Procedure

To delete an adapter:

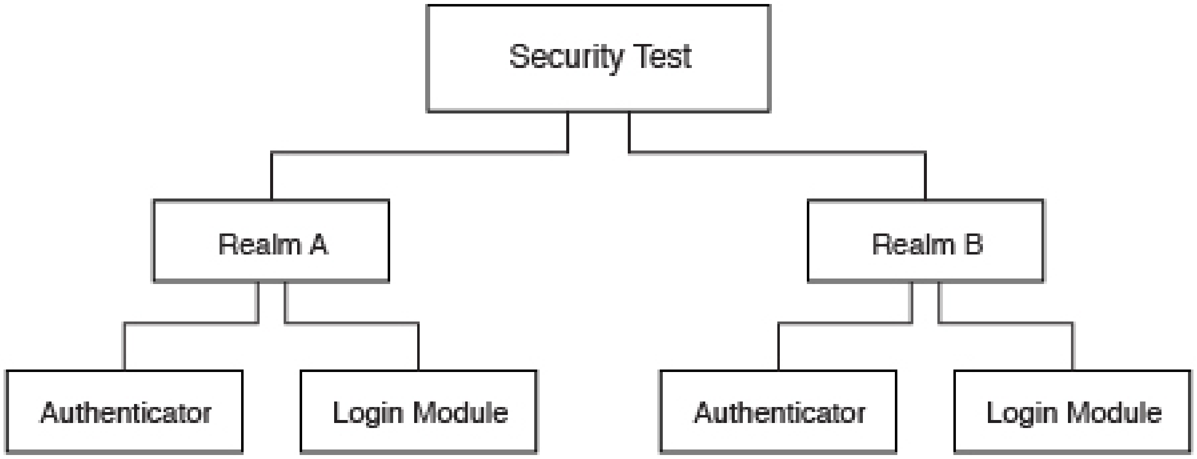
In the adapter list, click the trash icon in the adapter row.

MobileFirst security overview

IBM MobileFirst Platform Foundation for iOS has comprehensive support for various authentication and authorization methods.

MobileFirst 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 8-223.

Realm A realm creates a relationship between a MobileFirst login module and a MobileFirst authenticator to provide a means of authentication. For more information about realms, see "Authentication realms" on page 8-227.

Authenticator
An authenticator parses incoming requests from a MobileFirst 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 8-252.

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 8-252.

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 in a MobileFirst 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.

MobileFirst security configuration

For MobileFirst Server to protect a resource, such as an adapter procedure or an application environment, the administrator must first configure the MobileFirst Server instance.

Defining a login module

A login module is the most basic security element in the MobileFirst 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"
              audit="true
              expirationInSeconds="-1">
    <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 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

You can define a security test 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 MobileFirst app to handle any custom challenges.

For more information about challenge handlers, see the tutorials on the following Getting Started pages of the Developer Center: Custom Authentication and Custom Authentication in native iOS applications.

MobileFirst application environment protection

After a security test is configured with the appropriate realms, you can protect any resource. One option is to completely protect an application's environment 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:

```
<iPhone version="1.0" securityTest="BasicRequirements">
...
</iPhone>
```

This definition requires every iPhone device that connects to the server through your application to log in to the BasicRequirements security test.

MobileFirst adapter procedure protection

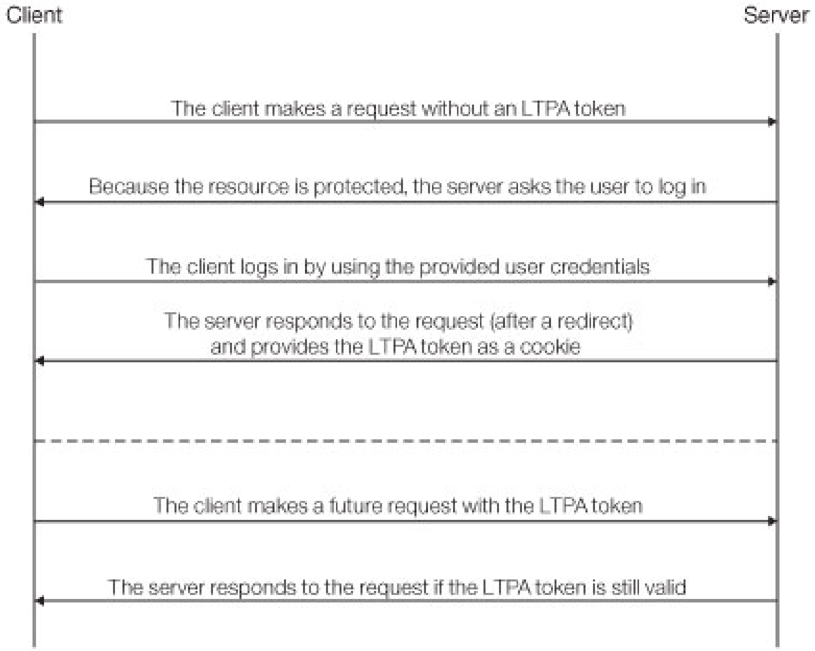
Another option is to protect a MobileFirst adapter procedure. Using the same security test, you can protect an adapter procedure. When the procedure is called and the user is not already authenticated into the security test, the client is required to authenticate. If you have an adapter procedure named 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" />
```

MobileFirst Security and LTPA

A lightweight third-party authentication (LTPA) token is a type of security token 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 back-end services. It can also be used as a single sign-on (SSO) token between the user and multiple servers.

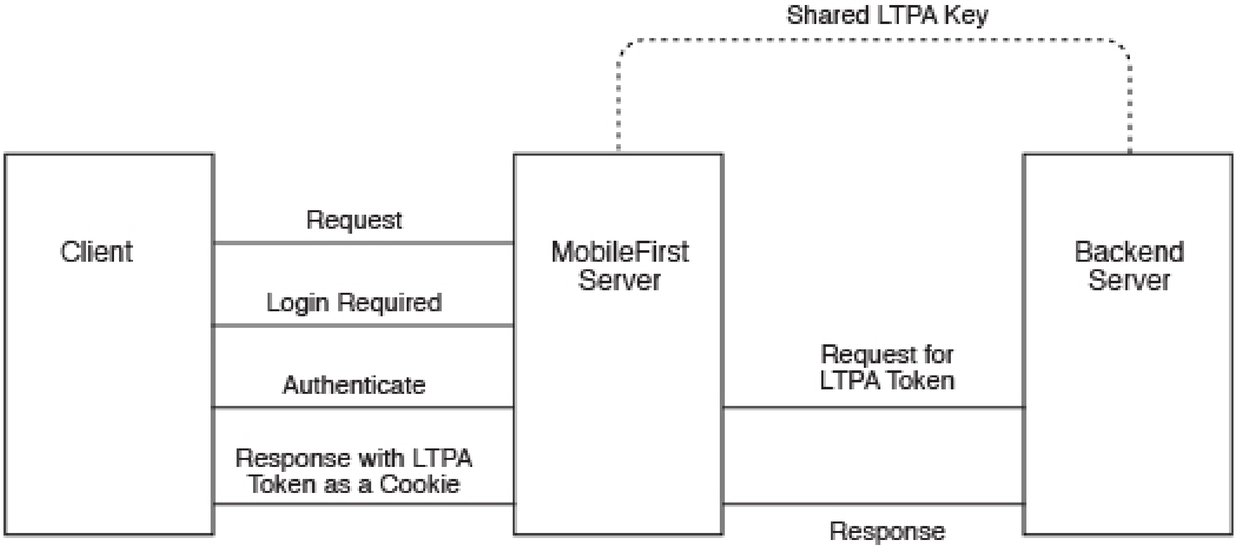
The following image shows a simple client <-> server flow with LTPA:


Client
Server
The client makes a request without an LTPA token
Because the resource is protected, the server asks the user to log in
The client logs in by using the provided user credentials
The server responds to the request (after a redirect) and provides the LTPA token as a cookie
The client makes a future request with the LTPA token
The server responds to the request if the LTPA token is still valid

After a user logs in, the server generates an LTPA token, which is an encrypted hash that contains authenticated user information. The token is signed by a private key that is shared among all the servers that want to decode it. The token is usually in cookie form for HTTP services. By sending the token as a cookie, there is no need for subsequent user interaction.

LTPA tokens have a configurable expiration time to reduce the possibility for session hijacking.

The following image shows a client <-> MobileFirst Server <-> back-end server flow with LTPA:



Your infrastructure can also use the LTPA token to communicate with a back-end server to act on behalf of the user. The user cannot directly access the back-end server. Enterprise environments should use a reverse proxy, such as DataPower or IBM Security Access Manager, in the DMZ, and place the MobileFirst Server in the intranet. This configuration ensures that access to the MobileFirst Server cannot be obtained until a user authenticates. For more information, see “Reverse proxy with LTPA” on page 11-97.

Configuring the MobileFirst LTPA realm:

The IBM MobileFirst Platform 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 MobileFirst 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 MobileFirst Server WAR file. The following example shows a sample login.html file:

   ```
   <html>
     <head>
       <title>Login</title>
     </head>
     <body>
       <form method="post" action="j_security_check">
         <input type="text"
                id="j_username"
                name="j_username"
                placeholder="User name" />
         <input type="password"
                id="j_password"
                name="j_password"
                placeholder="Password" />
         <input type="submit" id="login" name="login" value="Log In" />
       </form>
     </body>
   </html>
   ```

   The following example shows a sample loginError.html file:

   ```
   <html>
     <head>
       <title>Login Error</title>
     </head>
     <body>
       An error occurred while trying to log in.
     </body>
   </html>
   ```

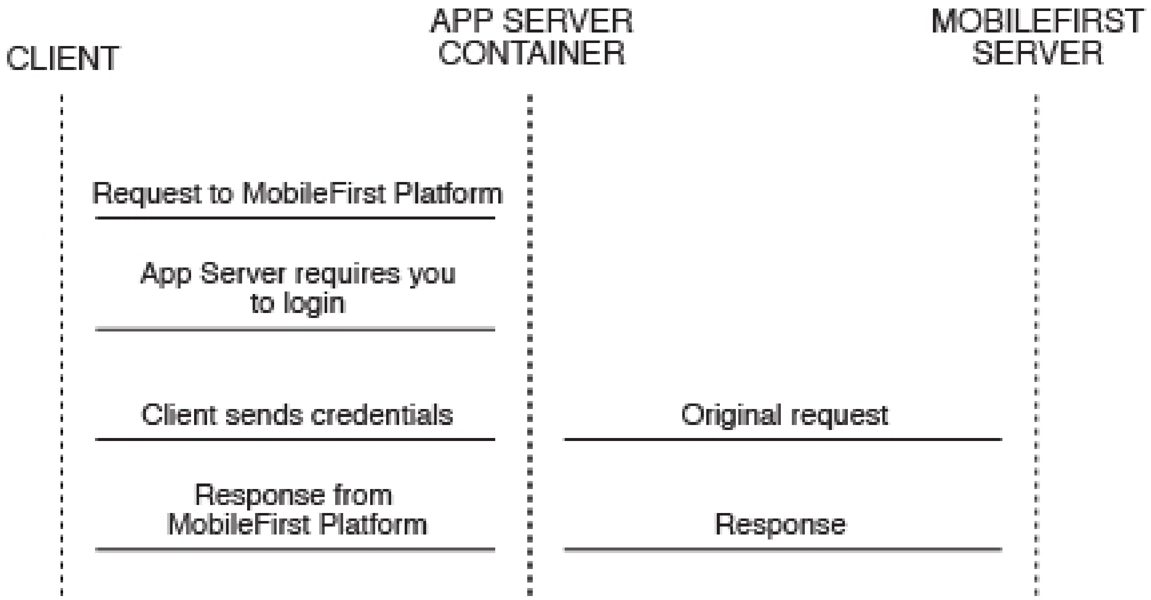
Supported configurations for LTPA:

IBM MobileFirst Platform Foundation for iOS 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 WebSphere LTPA-based authentication tutorial of the Developer Center. The application server is configured to protect all resources in the MobileFirst 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 MobileFirst 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:


CLIENT
APP SERVER CONTAINER
MOBILEFIRST SERVER
Request to MobileFirst Platform
App Server requires you to login
Client sends credentials
Original request
Response from MobileFirst Platform
Response

This option is not preferred for new configurations. The application server such as the WebSphere Application Server Liberty (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 MobileFirst 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:

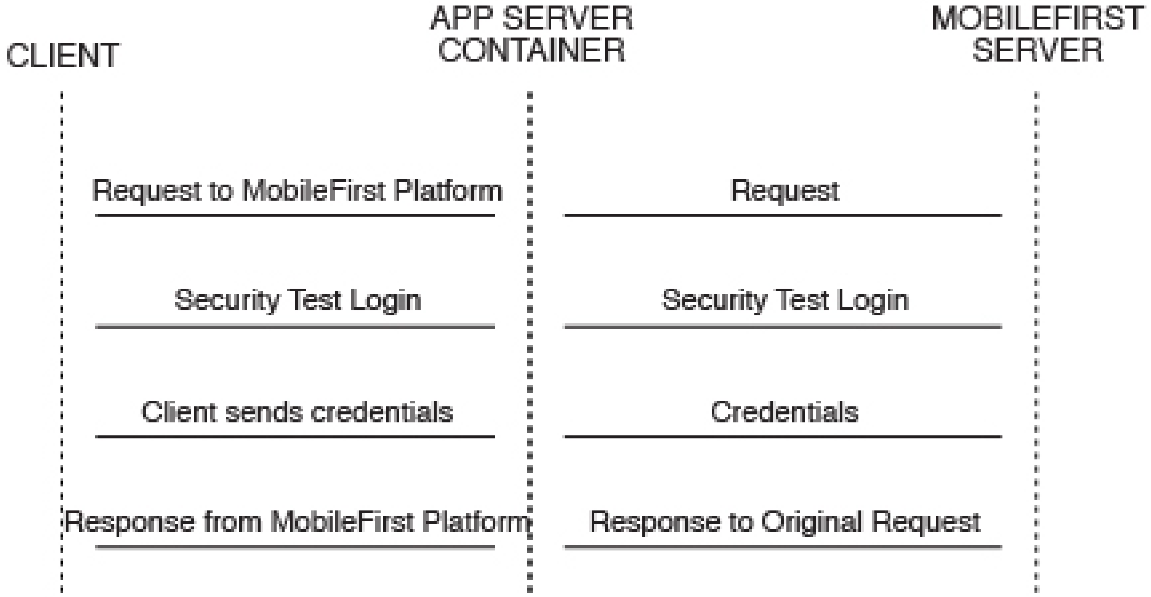
```
<featureManager>
  <feature>appSecurity-1.0</feature>
  <!-- Any additional features you need go here -->
</featureManager>

<!-- 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 MobileFirst Server WAR file to provide a way for the user to log in. For more information, see step 4 of “Configuring the MobileFirst LTPA realm” on page 11-92.

Protective MobileFirst security test (Option 2)

An alternative configuration allows the server to use all of the MobileFirst 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 MobileFirst 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 MobileFirst Server WAR file to provide a way for the user to log in. For more information, see step 4 of "Configuring the MobileFirst LTPA realm" on page 11-92.

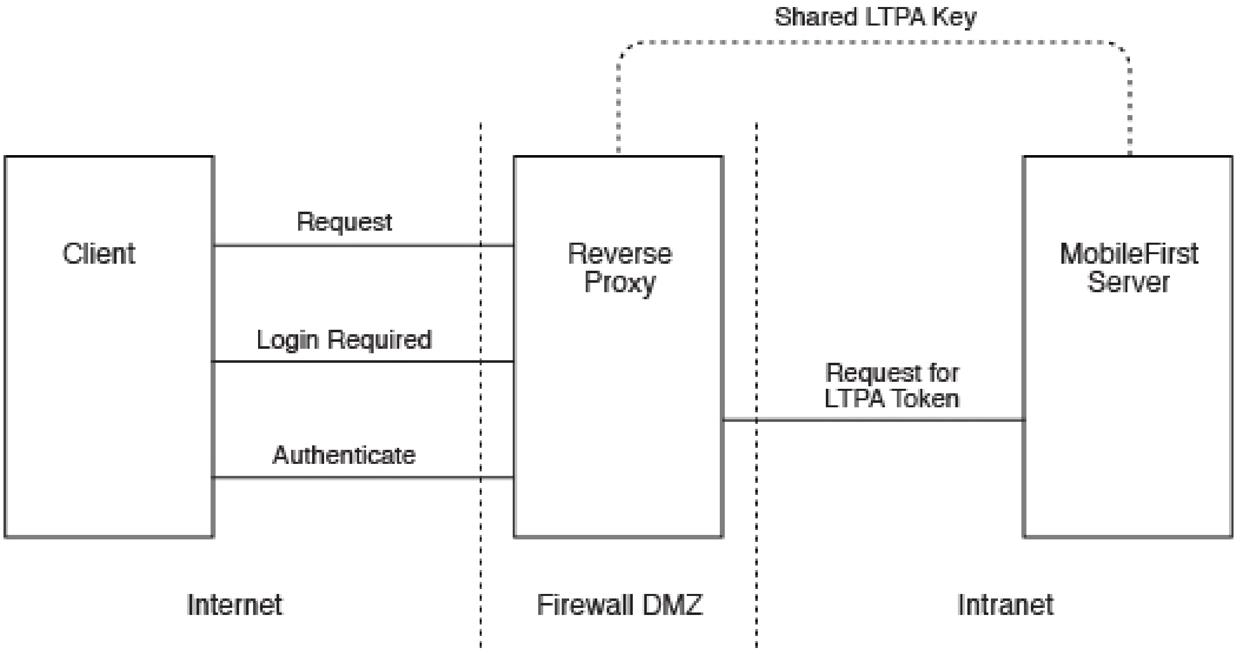
LTPA topologies and use cases:

IBM MobileFirst Platform Foundation for iOS supports various infrastructure topologies for a set of requirements that can take advantage of LTPA or MobileFirst 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 IBM MobileFirst Platform Foundation for iOS from handling vital user credentials or to use an existing authentication setup. The MobileFirst Server must be configured for LTPA authentication to get the user identity. Both supported LTPA configurations log the user in automatically if the LTPA token is valid and the user is authorized. For more information about integrating IBM MobileFirst Platform Foundation for iOS with a reverse proxy, see "Integration and authentication with a reverse proxy" on page 14-3.

The following image shows a reverse proxy flow:



Related information

Integration and authentication with a reverse proxy
You can use a reverse proxy to enable enterprise connectivity within a MobileFirst environment and to provide authentication to IBM MobileFirst Platform Foundation for iOS.

Integrating with DataPower as a security gateway and reverse proxy
Learn how to integrate with IBM WebSphere DataPower as a reverse proxy and security gateway.

Configuring the MobileFirst Server for Trusteer

Configure the IBM MobileFirst Platform 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 MobileFirst Trusteer realm.

Note: If code that uses the Trusteer realm accesses a resource that is protected by OAuth authentication, and the client has a valid token, the MobileFirst Server is not called. As a result, the server does not check whether the device passes all of the configured Trusteer parameters. The MobileFirst Server is called when the token expires or when the Trusteer realm inside the token expires. When the realm expires, the Trusteer authenticator is invoked and the server checks whether the devices passed all of the configured Trusteer parameters. For more information about the OAuth security model, see "OAuth-based security model" on page 8-188. For a general overview of the MobileFirst security configuration, see "MobileFirst security overview" on page 11-87 and "MobileFirst security configuration" on page 11-89.

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 11-18.

Table 11-18. 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 11-19 on page 11-99.

Table 11-19. Trusteer error codes.

Code	Description	Corresponding parameter
TAS_ROOT		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
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 completely with that security test by adding the `securityTest` attribute to the environment's element in the `authenticationConfig.xml` file:

```
<iPhone version="1.0" securityTest="TrusteerTest">
  ...
</iPhone>
```

   This definition requires every iPhone 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 MobileFirst 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 MobileFirst app to handle Trusteer challenges. The following samples are samples of simple challenge handlers:

   JavaScript

```
var trusteerChallengeHandler = WL.Client.createWLChallengeHandler("wl_basicTrusteerFraudDetectionRealm");

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 (reason code: " + error.reason + ")",
  [{text:"OK"}]);
};


//In case authenticator succeeds, there may still be alerts that developer should notify the user about:

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,  [{text:"OK"}]);
  }
}
```

Java

```
public class TrusteerChallengeHandler extends WLChallengeHandler {

  private static Logger logger = Logger.getInstance(TrusteerChallengeHandler.class.getSimpleName());
  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 + "(" + error + ")");
        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

```
// 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 handleFailure.
@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 MobileFirst Server for Trusteer” on page 11-97.

The following code samples are 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
}
```

Advanced security features

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 8-265.

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.

High availability

High availability is provided through clustering, the ability to provide multiple MobileFirst Server instances acting together.

Multiple MobileFirst Server instances enable horizontal scaling of the software as well as the prevention of a single point of failure.

Clustering

The MobileFirst 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 MobileFirst 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 MobileFirst project `WAR` (and the MobileFirst Shared library). For more information, see the IBM WebSphere Application Server V8 user documentation.

MobileFirst Server can run on a VMware virtual machine. In such cases, one machine image is created and then deployed again and again.

IBM MobileFirst Platform Foundation for iOS is *stateful*. It caches session state within the server memory. The result is that if one MobileFirst Server is taken offline, active user sessions are lost and the client is asked to log on again.

Configuring the load balancer

You can use hardware-based or software-based load balancers.

If you do not want to use a hardware-based load balancer, you can use a simpler, software-based load balancer, or reverse proxy such as the Apache Tomcat web server. Any load balancer that can support the following features is adequate:

- Sticky session: load balancing settings should depend on session dependency:
  - When session-independent mode is on, (default) sticky session should be off.
  - When session-independent mode is off, sticky session should be on.

  Note: For more information, see “Session-independent mode” on page 8-108.
- Reverse proxy capabilities: Mandatory
- SSL Acceleration: Optional

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 IBM MobileFirst Platform 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 MobileFirst 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 instances of MobileFirst Server.
2. Install the MobileFirst Server.
3. Apply the project `WAR`.

Firewalls

Firewalls can be configured at various layers of the IBM MobileFirst Platform Foundation for iOS architecture.

Firewalls in front of a MobileFirst Server use the typical configuration.

Special attention must be given to a firewall layer between the IBM MobileFirst Platform Foundation for iOS servers and the IBM MobileFirst Platform Foundation for iOS database.

- MobileFirst 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 MobileFirst Server instances, the number of connections might be higher than usual.

Disaster Recovery Site

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS has such a site for some of its customers.

Within the site, IBM MobileFirst Platform Foundation for iOS provides redundancy at every level: compensating load balancers, multiple IBM MobileFirst Platform Foundation for iOS 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 11-20. Data mirroring

Component	Description	Mirror frequency
IBM MobileFirst Platform Foundation for iOS 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.

Table 11-20. Data mirroring (continued)

Component	Description	Mirror frequency
IBM MobileFirst Platform Foundation for iOS Software, customization, and content	Any change in IBM MobileFirst Platform Foundation for iOS 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 MobileFirst apps in production

There are general guidelines for upgrading your MobileFirst apps when they are already in production, on the Application Center or in app stores.

The general procedure for deploying your MobileFirst apps for the first time to MobileFirst Server and the Application Center is as follows (see “Deploying an application from development to a test or production environment” on page 11-1).

1. Build and test your app by using IBM MobileFirst Platform Foundation for iOS, and use either the MobileFirst Operations Console or the supplied Ant tasks to deploy its `.wlapp` files to MobileFirst Server and the Application Center.
2. Submit the generated device app files (such as `.ipa` for iOS) to their respective app stores (the Apple Store in this example).
3. 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. You can choose from 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. Increment the app version number.
2. Build and test your project and generate new `.wlapp` and `.ipa` files for it.
3. Deploy the new `.wlapp` files to MobileFirst Server.
4. Submit the new `.ipa` file to Apple Store.
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 12-5 and "Defining administrator messages from MobileFirst Operations Console in multiple languages" on page 12-5.

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 12-5 and "Defining administrator messages from MobileFirst Operations Console in multiple languages" on page 12-5.
2. Increment the app version number.
3. Build and test your project and generate new `.wlapp` and `.ipa` files for it.
4. Deploy the new `.wlapp` files to MobileFirst Server.
5. Submit the new `.ipa` file to Apple Store.
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 MobileFirst Operations Console, supplying a message and link to the new version. See "Locking an application" on page 12-3 and "Remotely disabling application connectivity" on page 12-3.

Note: If you disable the old app, it is no longer able to communicate with MobileFirst Server. Users can still start the app and work with it offline unless you force a server connection on app startup.

Application authenticity

This feature will not work properly for clients that were built with an older version of IBM MobileFirst Platform Foundation for iOS when the application deployed is of a new product version 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 by using the newer version of IBM MobileFirst Platform Foundation for iOS, as described above.
2. Increment the versions of the upgraded applications.
3. Deploy the new WAR file that was built.

4. Deploy the new applications to the server alongside the applications that were built with the old IBM MobileFirst Platform Foundation for iOS.
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.

For more information about application authenticity, see “MobileFirst application authenticity overview” on page 8-221.

Deploying to the cloud in an IBM Container

You can deploy MobileFirst applications to the cloud in IBM Containers. This type of deployment enables you to deploy what you have developed in IBM MobileFirst Platform Foundation for iOS to a cloud platform such as IBM Bluemix.

The IBM MobileFirst Platform Foundation for iOS container-related offerings use IBM Containers, which is hosted on Bluemix (IBM's cloud-hosting environment). A container is based on an image format and provides an execution environment within itself.

IBM MobileFirst Platform Foundation for iOS on IBM Containers

Using MobileFirst Platform Foundation on IBM Containers, you can run instances of MobileFirst Server and MobileFirst Operational Analytics in containers on the cloud (IBM Bluemix).

This offering enables you to build Docker images of your MobileFirst applications and runtimes and then run them using the underlying infrastructure of IBM Containers on Bluemix.

Supported operating systems include: Linux and Mac OS X.

MobileFirst Platform Foundation on IBM Containers package overview

The MobileFirst Platform Foundation on IBM Containers package contains the artifacts to create a MobileFirst Server container and a MobileFirst Operational Analytics container, and the components necessary for configuring and deploying them to Bluemix.

- The MobileFirst Server container contains the following product components:
  - IBM MobileFirst Platform Server
  - IBM MobileFirst Platform Operations Console
  - MobileFirst Data Proxy
- The MobileFirst Operational Analytics container contains the following product components:
  - IBM MobileFirst Platform Operational Analytics server
  - IBM MobileFirst Platform Operational Analytics console
- Additional components:
  - Liberty for Java runtime
  - Configuration files
  - Scripts to build and deploy the containers

You will customize your product components in the given containers before they are built and deployed to the IBM Containers service on Bluemix.

See also "Package structure and contents"

Note: The Reports database has been deprecated and does not work with IBM MobileFirst Platform Foundation for iOS on IBM Containers. Instead, use a MobileFirst Operational Analytics container. Ensure that the project WAR files in your containers do not have any artifacts or configurations related to the deprecated Reports database.

Using the IBM MobileFirst Platform Foundation Containers

The main steps for using MobileFirst Platform Foundation on IBM Containers are:

1. Customizing the product components included in the container.
2. Building the container image format with your customizations.
3. Deploying and running the built images on the IBM Containers service.

Prerequisites

- A Java Runtime is required. See the system requirements for version information.
- Cloud Foundry CLI plug-in for IBM Containers (**`cf ic`**).

  Note: The prerequisite for using this plug-in is to install Cloud Foundry CLI. For details, see CLI and dev tools.
- A Docker installation
- An IBM Bluemix account

Evaluation version on developerWorks

There is an evaluation version of MobileFirst Platform Foundation on IBM Containers V7.1 available as well as a related tutorial. You can download the evaluation package from IBM developerWorks.

Also on developerWorks, learn how you can use IBM UrbanCode™ Deploy to define a DevOps deployment solution for your MobileFirst Platform Foundation artifacts.

Package structure and contents

This topic contains important information about scripts and other contents of the MobileFirst Platform Foundation on IBM Containers package (`ibm-mfpf-container-`*`7.1.0.0`*).

The MobileFirst Platform Foundation on IBM Containers package provides a means to build your custom MobileFirst applications and deploy them to IBM Containers on Bluemix.

After you download the package, extract the contents to your development environment (to the `ibm-mfpf-container-`*`7.1.0.0`* folder, also referred to in this document as *package_root*). The following tables contain descriptions of the top-level folders of the package offering.

Table 11-21. Top-level folders

Folder	Description
dependencies	Contains the IBM MobileFirst Platform Foundation for iOS runtime and IBM Java JRE 7.1.

Table 11-21. Top-level folders (continued)

Folder	Description
mfpf-analytics	Contains the artifacts required to build and deploy the MobileFirst Operational Analytics container.
mfpf-libs	Contains MobileFirst product component libraries and CLI.
mfpf-server	Contains the artifacts required to build and deploy a MobileFirst Server container.

Table 11-22. MobileFirst Operational Analytics container and MobileFirst Server container folders. The sub-folder names within the MobileFirst Operational Analytics container (mfpf-analytics) folder and MobileFirst Server container (mfpf-server) folder

Folder	Description
scripts	Contains the properties files and scripts for building and deploying the containers. Other than the customizable `args/*.properties` files, do not modify any elements in this folder.
usr	Contains user-configurable elements, such as keystores, properties, registry, and projects. Elements in this folder can be modified but not deleted.

Scripts

The scripts provided build and run the MobileFirst Operational Analytics container image and MobileFirst Server container image as containers.

Scripts can only be run from within the `scripts` folder. Do not modify the given folder structure.

The following methods are supported for passing parameters to the scripts:

- Command-line arguments (Usage: *scriptname*`.sh [-`*command*`|--`*command*`]` *ARGUMENT*)
- Interactive method (By running the script with no command-line arguments.)
- Properties files (By customizing the related `args/*.properties` files.)

  You can find information about the required and optional arguments in the properties files, such as the default values, input descriptions, and so forth. The script properties files are located in the following folders in the *package_root/*:

  mfpf-analytics`/scripts/args`

  mfpf-server`/scripts/args`

  Example command execution usage: `prepareserver.sh args/prepareserver.properties`

For script usage help, use the **-h** or **--help** command-line arguments (for example, *scriptname*`.sh --help`).

Setting up MobileFirst Platform Foundation on IBM Containers

This topic contains an outline of the steps required to set up MobileFirst Platform Foundation on IBM Containers.

Procedure

1. Install the required programs and verify that you have met the requirements listed in the Prerequisites section.

2. Download the MobileFirst Platform Foundation on IBM Containers package and extract the contents to a folder in your development environment. (This folder is referred to as your installation directory or *package_root*.)
3. Set the namespace for the container image registry:
   - To get the existing namespace, run the command **- cf ic namespace get**
   - To set the namespace, run the command **- cf ic namespace set** "*your_namespace*"

   Learn more...
4. Add your MobileFirst projects.
5. Customize the product components in the container as needed (see "Customizing MobileFirst Server containers" on page 11-111).
6. Configure security.
7. Build the images:
   - If you are using MobileFirst Operational Analytics, build this image first and then deploy it to the IBM Containers service on Bluemix.
   - Build the MobileFirst Server container image and then deploy it to the IBM Containers service on Bluemix.
8. Deploy your customized container images to Bluemix.
9. Run the container.

MobileFirst Server containers

This section contains the information you need to configure and run MobileFirst Server containers.

Configuring a database instance on IBM Cloudant:

You can configure a IBM MobileFirst Platform Foundation for iOS container to use a database instance on Cloudant.com.

Before you begin

To configure your container to use a database instance that is hosted on Cloudant.com, you need your Cloudant user name and password and the URL of the database instance.

This type of database configuration does not require that you run the `mfpf-server/scripts/prepareserverdbs.sh` script.

Procedure

1. Using the example code snippet, create an XML configuration file for the MobileFirst Server administration database and place it in the `mfpf-server/usr/config` folder. Make sure to customize the example code with your Cloudant.com account information and URL.
   ```
   <?xml version="1.0" encoding="UTF-8"?><server description="new server">
   <jndiEntry jndiName="${env.MFPF_ADMIN_ROOT}/mfp.db.cloudant.username" value="cloudant_username"/>
   <jndiEntry jndiName="${env.MFPF_ADMIN_ROOT}/mfp.db.cloudant.password" value="cloudant_password"/>
   <jndiEntry jndiName="${env.MFPF_ADMIN_ROOT}/mfp.db.cloudant.url" value="cloudant_db_url"/>
   </server>
   ```
2. Using the example code snippet, create an XML runtime database configuration file for each runtime project to be deployed using MobileFirst Server and place

the files in the `mfpf-server/usr/config` folder. Make sure to customize the example code with your project name and your Cloudant.com account information and URL.

```
<?xml version="1.0" encoding="UTF-8"?><server description="new server">
<jndiEntry jndiName="<project_name/mfp.db.cloudant.username" value="<cloudant_username"/>
<jndiEntry jndiName="project_name/mfp.db.cloudant.password" value="cloudant_password"/>
<jndiEntry jndiName="project_name/mfp.db.cloudant.url" value="<cloudant_db_url"/>
<application id="project_name" location="project_name.war" name="project_name" type="war">
<classloader delegation="parentLast">
<privateLibrary id="worklightlib_project_name">
<fileset dir="${shared.resource.dir}/lib" includes="worklight-jee-library.jar"/>
<fileset dir="${wlp.install.dir}/lib" includes="com.ibm.ws.crypto.passwordutil*.jar"/>
</privateLibrary>
</classloader>
</application>
</server>
```

What to do next

After the XML configuration file is in place, continue with the container set up process. The next step is to run the `mfpf-server/scripts/prepareserver.sh` script. The XML file you created gets used by the script for database configuration.

See also:

"Building and running the MobileFirst Operational Analytics container" on page 11-126

"Building and running the MobileFirst Server container" on page 11-114

Customizing MobileFirst Server containers:

You must add projects and customize the related project components before building MobileFirst Server container images.

Custom applications built using MobileFirst Studio, or using Xcode and the MobileFirst Platform Command Line Interface for iOS can be deployed in a MobileFirst Server container.

The container gets created from the artifacts that are provided in the MobileFirst Platform Foundation on IBM Containers package. For an overview of the package contents and folder structure, see "Package structure and contents" on page 11-108.

The customizable elements for the MobileFirst Server container are located in *package_root*/`mfpf-server/usr`. The following tables describe the sub folders and files to use for customization.

Table 11-23. Descriptions of the `mfpf-server/` sub folders

Folder	Description
`./usr`	Contains the customization template for the MobileFirst Server container.
`./usr/bin`	Contains the script file (mfp-init) that gets executed when the container starts. You can add custom code to the script, however, do not modify the existing code.

Table 11-23. Descriptions of the `mfpf-server/` sub folders (continued)

Folder	Description
`./usr/config`	Contains the server configuration fragments (keystore, server properties, user registry) used by MobileFirst Server. • `keystore.xml` - the configuration of the repository of security certificates used for SSL encryption. The files listed must be referenced in the `./usr/security` folder. • `mfpfproperties.xml` - configuration properties for the MobileFirst Server. See the supported properties listed in these topics: "List of JNDI properties for MobileFirst Server administration" on page 6-101 "Configuration of MobileFirst applications on the server" on page 11-50 • `registry.xml` - user registry configuration. The basicRegistry (a basic XML-based user-registry configuration is provided as the default. User names and passwords can be configured for basicRegistry or you can configure ldapRegistry.
`./usr/env`	Contains the environment properties used for server initialization (`server.env`) and custom JVM options `jvm.options`. See Table 11-24 for a list of supported server environment properties.
`./usr/jre-security`	Add JRE security-related files (such as the JRE truststore, policy .jar files, and so forth) to be updated on the container. The files in this folder get copied to the *`JAVA_HOME`*`/jre/lib/security/` folder in the container.
`./usr/projects`	Contains the existing projects that have been deployed to MobileFirst Server. You can add MobileFirst project WAR files or add the entire folder structure of a M
`./usr/projects/`*`project_name`*	Contains the project to deploy to MobileFirst Server.
`./usr/projects/`*`project_name`*`/bin`	Contains the project WAR file. The name of the project and the name of the project WAR file must be the same.
`./usr/projects/`*`project_name`*`/server/conf`	Contains `authenticationConfig.xml` and `SMSConfig.xml` . These settings override the settings provided in the project WAR file.
`./usr/projects/`*`project_name`*`/server/lib`	Contains the library (JAR) files that are required by the project.
`./usr/security`	Contains your keystore, truststore, and LTPA keys (`ltpa.keys`) files.
`./usr/ssh`	Contains the ssh public key file (`id_rsa.pub`) to enable ssh on the container.
`./usr/wxs`	Contains The WebSphere eXtreme Scale client library when the IBM Data Cache service on Bluemix is used as the attribute store for MobileFirst Server.

Table 11-24. Supported server environment properties (`server.env`)

Property	Default Value	Description
MFPF_SERVER_HTTPPORT	9080*	The port used for client HTTP requests. Use -1 to disable this port.

Table 11-24. Supported server environment properties (`server.env`) (continued)

Property	Default Value	Description
MFPF_SERVER_HTTPSPORT	9443*	The port used for client HTTP requests secured with SSL (HTTPS). Use -1 to disable this port.
MFPF_CLUSTER_MODE	Standalone	Configuration not required. Valid values are Standalone or Farm. The value Farm is automatically set when the container is run as a container group.
MFPF_ADMIN_ROOT	worklightadmin	The context root at which the MobileFirst Server Administration Services are made available.
MFPF_CONSOLE_ROOT	worklightconsole	The context root at which the MobileFirst Operations Console is made available.
MFPF_ADMIN_GROUP	worklightadmingroup	The name of the user group possessing the predefined role *worklightadmin*.
MFPF_DEPLOYER_GROUP	worklightdeployergroup	The name of the user group possessing the predefined role *worklightdeployer*.
MFPF_MONITOR_GROUP	worklightmonitorgroup	The name of the user group possessing the predefined role *worklightmonitor*.
MFPF_OPERATOR_GROUP	worklightoperatorgroup	The name of the user group possessing the predefined role *worklightoperator*.
MFPF_SERVER_ADMIN_USER	WorklightRESTUser	The Liberty server administrator user for MobileFirst Server Administration Services.
MFPF_SERVER_ADMIN_PASSWORD	worklightadmin Ensure that you change the default value to a private password before deploying to a production environment.	The password of the Liberty server administrator user for MobileFirst Server Administration Services.
MFPF_ADMIN_USER	admin	The user name for the administrator role for MobileFirst Server operations.
MFPF_ADMIN_PASSWORD	admin	The password for the administrator role for MobileFirst Server operations.
publicKeyServerUrl		The URL to the MobileFirst runtime that runs the mobile apps with MobileFirst Data Proxy.

*Do not modify the default port number. Read more in the following section.

After you finish customizing an image, it is ready to be built and run on IBM Containers for Bluemix.

Important: If you are going to use MobileFirst Operational Analytics, you must build and run the MobileFirst Operational Analytics container before deploying and running the MobileFirst Server container.

Containers must be restarted after any configuration changes have been made (`cf ic restart` *`containerId`*). For container groups, you must restart each container instance within the group. For example, if a root certificate changes, each container instance must be restarted after the new certificate has been added.

Port number limitation

There is currently an IBM Containers limitation with the port numbers that are available for public domain. Therefore, the default port numbers given for the MobileFirst Operational Analytics container and the MobileFirst Server container (9080 for HTTP and 9443 for HTTPS) cannot be altered. Containers in a container group must use HTTP port 9080. Container groups do not support the use of multiple port numbers or HTTPS requests.

Building and running the MobileFirst Server container:

Use the scripts that are provided to build and run your customized image. You can find the scripts in the MobileFirst Platform Foundation on IBM Containers package installation directory under `mfpf-server/scripts`.

Before you begin

- You have finished customizing the image.

About this task

Scripts can only be run from within the `scripts` folder. Do not modify the given folder structure.

The following methods are supported for passing parameters to the scripts:

- Command-line arguments (Usage: *`scriptname`*`.sh [-`*`command`*`|--`*`command`*`]` *`ARGUMENT`*)
- Interactive method (By running the script with no command-line arguments.)
- Properties files (By customizing the related `args/*.properties` files.)

  You can find information about the required and optional arguments in the properties files, such as the default values, input descriptions, and so forth. The script properties files are located in the following folders in the *package_root/*:

  `mfpf-analytics``/scripts/args`

  `mfpf-server``/scripts/args`

  Example command execution usage: `prepareserver.sh args/prepareserver.properties`

Procedure

The first step describes how to retrieve the public IP address to be bound with the container, which is a required argument when you run the `startserver.sh` script (as described in step 2).

1. Retrieve and take note of a public IP address to bind to the container.

   To get IP information, use Cloud Foundry CLI plug-in for IBM Containers (**`cf ic`**) commands.

   - To retrieve the list of IP addresses that are potentially available to you (based on user ID), run **`cf ic ip list`**.

     The IP addresses that are listed with no corresponding container ID are available for use.

     An IP address with a corresponding container ID indicates that the IP address is already in use. If all IP addresses are already in use, you can request a new IP address.

   - To request a new IP address, run **`cf ic ip request`**.

2. Create a database service.

   Important: Complete this step before you run the **`prepareserverdbs.sh`**, described in the next step.

   You can create a database service instance on Bluemix in two ways:

   - Using the Bluemix dashboard.
   - Using the Cloud Foundry command line utility.

   Use one of the following methods to create a database service instance.

   Using the Bluemix dashboard:

   To create a service instance of dashDB, Cloudant NoSQL DB or SQL Database on Bluemix, follow the next steps.

   a. Log in to Bluemix.

   b. Select the space name where you want to create the service instance (example: dev).

   c. Select **Data & Analytics** category.

   d. Click on the **Get started now!** icon, to go to the catalog for services under the **Data & Analytics** category.

   e. From the services catalog search for dashDB, Cloudant NoSQL DB or SQL Database.

   f. Click the desired service from the search result.

   g. Select a value of `Leave unbound` for the **App** field. Enter a name for the service instance in the **`Service name`** field. Select a suitable **`Plan`** for the service and click **CREATE**.

      Remember: If you choose to create dashDB service instance then select a suitable service **`Plan`**, which supports Online Transaction Processing (OLTP) workloads, in step g.

   Using the Cloud Foundry command line utility:

   a. Log in to Bluemix by using the following command:

      ```
      cf login [-a API_URL] [-u USERNAME] [-p PASSWORD] [-o ORG] [-s SPACE]
      ```

      Where:

      `-u user_name`
      Your user name. This parameter is optional.

-p password
Your password.

Security consideration: If you provide the password using the **-p** parameter, the password might be recorded in your command line history. If you do not want the password to be recorded, instead of using the **-p** parameter, consider entering the password when the command line interface prompts you, during the execution of the **cf login** command.

-o organization_name
The name of the organization that you want to log in to.

-s space_name
The name of the space that you want to log in to.

-a https://api.DomainName
The URL of the API endpoint of Bluemix. This parameter is optional.

b. To create the service instance, run the following command:

```
cf create-service service_name plan_name service_instance
```

You can use one of the following examples:

```
cf create-service dashDB EnterpriseTransactional2.8.500
mfpfdashdbservice
```

```
cf create-service dashDB EnterpriseTransactional12.128.1400
mfpfdashdbservice
```

```
cf create-service cloudantNoSQLDB Shared mfpfcloudantservice
```

```
cf create-service sqldb sqldb_premium mfpfsqldbservice
```

Note: The dashDB service instance can take a while before it's provisioning is completed and it is made available for use.

3. Run the scripts in the order listed:

initenv.sh
This script logs in to the container service. You must run this script before you can run any subsequent scripts.

Your Bluemix log-in credentials as well as the organization name and space name are required arguments.

prepareserverdbs.sh
This script must be run once for the administrative database and once for every project or every run time.

Your Bluemix database service instance name is a required argument.

Important: If you use Cloudant NoSQL DB to store administration and runtime data, then do not use the Shared plan. Select one of the Cloudant NoSQL DB enterprise service plans, which provides the Cloudant cluster on dedicated hardware. If you use the dashDB service to store administration and runtime data, then you need to choose only a dashDB service plan that supports OLTP (Online Transaction Processing) workloads. Other dashDB service plans are not supported.

If you are configuring a runtime database, the name of the runtime project to be deployed on the MobileFirst Server is required.

You can optionally specify a database schema name. For an administrative database, the default schema name is WLADMIN. For a runtime database, the default schema name is the runtime name.

Note: The Reports database has been deprecated and does not work with IBM MobileFirst Platform Foundation for iOS on IBM Containers. Instead, use a MobileFirst Operational Analytics container. Ensure that the project WAR files in your containers do not have any artifacts or configurations related to the deprecated Reports database.

`prepareserver.sh`

This script builds the server image with the mfp-server customizations and sends the image to IBM Containers.

The MobileFirst Server image name is a required argument. Use the following format: *BluemixRegistry/PrivateNamespace/ImageName*:*TagName*. Example: *registry.ng.bluemix.net/PrivateNamespace/mfpserver71*

`startserver.sh`

The script runs the MobileFirst Server image as a stand-alone container.

The MobileFirst Server image name, the container name, and the public IP address from which the container is started (from step 1) are required arguments for running the image as a container.

Tip: If you are running a **`startserver.sh`** or **`startservergroup.sh`** script interactively and configuring an analytics image (by using MFPF_PROPERTIES), you must provide the configuration information every time the script is run, and for each runtime, to avoid losing the configuration. For example, if you provided an analytics configuration (such as `MFPF_PROPERTIES=`*`wl.analytics.url:http://127.0.0.1/analytics-service/v2`*`,`*`wl.analytics.console.url:http://127.0.0.1/analytics/consoleproperties`* for the first runtime but did not provide it the next time that you ran the script for a different runtime, the configuration for the MobileFirst Server would be lost.

`startservergroup.sh`

The script runs the MobileFirst Server image as a container group.

Required arguments include: the MobileFirst Server image name, the container group name, the minimum and maximum number of container instances within the group, and the host name to which the group must be mapped.

Adding, updating, and removing projects:

This topic provides instructions for adding a project to and updating an existing project in a container, and for removing a project from an image.

Adding a project to your IBM MobileFirst Platform Foundation for iOS container

1. Run the **`prepareserverdbs.sh`** script by specifying the new project name. This will create the project-related schema in the database.
2. Add the new project to the `ibm-mfpf-container-`*`7.1.0.0`*`/mfp-server/projects` folder.
3. Run the **`prepareserver.sh`** script to rebuild the server image and push it to the IBM Containers service.

4. Run the script **startserver.sh** to run the image as a standalone container or **startservergroup.sh** to run the image as a container group.

Updating a project in your IBM MobileFirst Platform Foundation for iOS container

1. Update the artifacts that need to be changed in the ibm-mfpf-container-*7.1.0.0*/mfp-server/projects/*project_name* folder.
2. Run the **prepareserver.sh** script to rebuild the server image and push it to the IBM Containers service.
3. Run the **startserver.sh** script to run the server image as a standalone container or **startservergroup.sh** to run the server image as a container group.

Removing a project from the IBM MobileFirst Platform Foundation for iOS container image

1. Run the **removeproject.sh** script to remove the project from the IBM MobileFirst Platform Foundation for iOS container image. You can also choose to clear the project-related data from the databases.
2. After removing the project from the project folder, run the **prepareserver.sh** script to rebuild the server image and push it to the IBM Containers service.
3. Run the **startserver.sh** script to run the server image as a standalone container or **startservergroup.sh** to run the server image as a container group.

Configuring session-independent mode:

You can configure-session independent mode on MobileFirst Server.

Before you begin

You must use the IBM Data Cache service on Bluemix to configure the MobileFirst Server in session-independent mode. Learn more...

Procedure

Use the following steps to configure MobileFirst Server to use IBM Data Cache service as the attribute store.

1. Create an IBM Data Cache service instance on Bluemix and bind the service to a new or existing application.
2. Download WebSphere eXtreme Scale for Developers - Liberty Profile (wxs-wlp_*vnnn*.jar) and place it in the *package_root*/mfpf-server/usr/wxs folder.
3. Add the following code to the .\usr\config\mfpfproperties.xml file.

   ```
   <featureManager>
     <feature>eXtremeScale.client-1.1</feature>
   </featureManager>
   <jndiEntry jndiName="project_name/mfp.session.independent" value="true"/>
   <jndiEntry jndiName="project_name/mfp.attrStore.type" value="datacache"/>
   ```

   The *project_name* value is the name of the IBM MobileFirst Platform Foundation for iOS project for which the IBM Data Cache service must be enabled as the attribute store.
4. To bind the IBM Data Cache service app to the container, use the **startserver.sh** command to start the MobileFirst Server container and then provide the application name (from step 1).

Script overview and usage:

Use the scripts for configuring, building, and deploying a MobileFirst Server container image.

The scripts are located in the `package_root/mfpf-server/scripts` folder.

initenv.sh script:

This script file creates the environment for building and running a MobileFirst Server container and performs the tasks necessary for logging into Bluemix and the IBM Containers environment. Run this script before running any other scripts for building and deploying IBM MobileFirst Platform Foundation for iOS container images.

Table 11-25. Mandatory command-line arguments

Command-line argument	Description
[**-u**|**--user**] *BLUEMIX_USER*	Bluemix user ID or email address
[**-p**|**--password**] *BLUEMIX_PASSWORD*	Bluemix password
[**-o**|**--org**] *BLUEMIX_ORG*	Bluemix organization name
[**-s**|**--space**] *BLUEMIX_SPACE*	Bluemix space name

Usage example:

```
initenv.sh
  --user Bluemix_user_ID
  --password Bluemix_password
  --org Bluemix_organization_name
  --space Bluemix_space_name
```

Table 11-26. Optional command-line arguments

Command-line argument	Description
[**-a**|**--api**] *BLUEMIX_API_URL*	Bluemix API endpoint (Defaults to `https://api.ng.bluemix.net`)

prepareserverdbs.sh script:

This script configures MobileFirst Server databases (administration and runtime). This script must be run once to configure the administration database and then once for each runtime.

Table 11-27. Mandatory command-line arguments

Command-line argument	Description
[**-n**|**--name**] *DB_SRV_NAME*	Bluemix database service instance name Supported database service types are dashDB, Cloudant DB and SQL Database.

Usage example:

```
prepareserverdbs.sh
 --type DB_TYPE sqldb
 --name DB_SRV_NAME database_name
 --plan DB_SRV_PLAN sqldb_small
 --appname APP_NAME Bluemix_app_name
```

Table 11-28. Optional command-line arguments

Command-line argument	Description
[**-r**|**--runtime**] *RUNTIME_NAME*	MobileFirst runtime name (Required for configuring runtime databases only.)
[**-sn**|**--schema**] *SCHEMA_NAME*	Database schema name (Defaults to WLADMIN for administration databases or the

prepareserver.sh script:

This script creates the MobileFirst Server image and pushes it to IBM Containers on Bluemix. Ensure that you have run the `prepareserverdbs.sh` to configure the database before running this script.

Table 11-29. Mandatory command line arguments

Command line argument	Description
[**-t**|**--tag**] *SERVER_IMAGE_NAME*	Name to be used for the customized MobileFirst Server image. Format: *registryUrl/namespace/imagename*

Usage example:

```
prepareserver.sh --tag SERVER_IMAGE_NAME registryUrl/namespace/imagename
```

Table 11-30. Optional command line arguments

Command line argument	Description
[**-l**|**--loc**] *PROJECT_LOC*	The location of the MobileFirst project. Multiple project locations can be delimited by commas.

startserver.sh script:

This script runs the MobileFirst Server image as a container on IBM Containers on Bluemix. Ensure that you have run the `prepareserver.sh` script to upload the image to the IBM Containers registry before running this script.

Table 11-31. Mandatory command line arguments

Command line argument	Description
[**-t**|**--tag**] **SERVER_IMAGE_TAG**	Name of the MobileFirst Server image.
[**-n**|**--name**] **SERVER_CONTAINER_NAME**	Name of the MobileFirst Server container
[**-i**|**--ip**] **SERVER_IP**	IP address that the MobileFirst Server container should be bound to. (You can provide an available public IP or request one using the **cf ic ip request** command.)

Usage example:

```
startserver.sh
 --tag image_tag_name
 --name container_name
 --ip container_ip_address
```

Table 11-32. Optional command line arguments

Command line argument	Description
[**-an**|**--appName**] **APP_NAME**	Bluemix application name that should be bound to the container
[**-h**|**--http**] *EXPOSE_HTTP*	Expose HTTP Port. Accepted values are Y (default) or N.
[**-s**|**--https**] *EXPOSE_HTTPS*	Expose HTTPS Port. Accepted values are Y (default) or N.
[**-m**|**--memory**] *SERVER_MEM*	Assign a memory size limit to the container in megabytes (MB). Accepted values are 1024 MB (default) and 2048 MB.
[**-se**|**--ssh**] *SSH_ENABLE*	Enable SSH for the container. Accepted values are Y (default) or N.
[**-sk**|**--sshkey**] *SSH_KEY*	The SSH Key to be injected into the container. (Provide the contents of your `id_rsa.pub` file.)
[**-tr**|**--trace**] *TRACE_SPEC*	The trace specification to be applied. Default: **=info*
[**-ml**|**--maxlog**] *MAX_LOG_FILES*	The maximum number of log files to maintain before they are overwritten. The default is 5 files.
[**-ms**|**--maxlogsize**] *MAX_LOG_FILE_SIZE*	The maximum size of a log file. The default size is 20 MB.
[**-v**|**--volume**] *ENABLE_VOLUME*	Enable mounting volume for container logs. Accepted values are Y or N (default).
[**-e**|**--env**] *MFPF_PROPERTIES*	Specify MobileFirst properties as comma-separated key:value pairs. Example: *wl.analytics.url:http://127.0.0.1/analytics-service/v2,wl.analytics.console.url:http://127.0.0.1/analytics/console* Note: If you specify properties using this script, ensure t

startservergroup.sh script:

This script runs a MobileFirst Server image as a container group on IBM Containers on Bluemix. Before running `startservergroup.sh`, ensure that you have run the `prepareserver.sh` script to upload the container image to the IBM Containers registry.

Table 11-33. Mandatory command-line arguments

Command-line argument	Description
[**-t**|**--tag**] *SERVER_IMAGE_TAG*	The name of the MobileFirst Server container image in the Bluemix registry.
[**-gn**|**--name**] *SERVER_CONTAINER_NAME*	The name of the MobileFirst Server container group.

Table 11-33. Mandatory command-line arguments (continued)

Command-line argument	Description
[**-gh**|**--host**] *SERVER_CONTAINER_GROUP_HOST*	The host name of the route.
[**-gs**|**--domain**] *SERVER_CONTAINER_GROUP_DOMAIN*	The domain name of the route.

Usage example:

```
startservergroup.sh
 --tag image_name
 --name container_group_name
 --host container_group_host_name
 --domain container_group_domain_name
```

Table 11-34. Optional command-line arguments

Command-line argument	Description
[**-gm**|**--min**] **SERVERS_CONTAINER_GROUP_MIN**	The minimum number of container instances. The default value is 1.
[**-gx**|**--max**] *SERVER_CONTAINER_GROUP_MAX*	The maximum number of container instances. The default value is 1.
[**-gd**|**--desired**] *SERVER_CONTAINER_GROUP_DESIRED*	The desired number of container instances. The default value is 2.
[**-a**|**--auto**] *ENABLE_AUTORECOVERY*	Enable the automatic recovery option for the container instances. Accepted values are Y or N (default).
[**-an**|**--appName**] *APP_NAME*	The name of the Bluemix app to be bound to the container instance.
[**-tr**|**--trace**] *TRACE_SPEC*	The trace specification to be applied. Default: **=info*
[**-ml**|**--maxlog**] *MAX_LOG_FILES*	The maximum number of log files to maintain before they are overwritten. The default is 5 files.
[**-ms**|**--maxlogsize**] *MAX_LOG_FILE_SIZE*	The maximum size of a log file. The default size is 20 MB.
[**-e**|**--env**] *MFPF_PROPERTIES*	Specify MobileFirst properties as comma-separated key:value pairs. Example: *wl.analytics.url:http://127.0.0.1/analytics-service/v2*
[**-m**|**--memory**] *SERVER_MEM*	Assign a memory size limit to the container in megabytes (MB). Accepted values are 1024 MB (default) and 2048 MB.
[**-v**|**--volume**] *ENABLE_VOLUME*	Enable mounting volume for container logs. Accepted values are Y or N (default).

Learn more about creating scalable container groups.

removeproject.sh script:

This script removes a project from the IBM MobileFirst Platform Foundation for iOS configuration.

After removing a project, you must run the `prepareserver.sh` script to update the image.

Table 11-35. Mandatory command-line arguments

Command-line argument	**Description**
[**-r**|**--runtime**] *RUNTIME_NAME*	The name of the run time to be removed.
[**-i**|**--ip**] *SERVER_IP*	The IP address or route to the MobileFirst Server container.
[**-p**|**--port**] *SERVER_PORT*	The HTTPS port number exposed on the MobileFirst Server container.
[**-lu**|**--libertyadminusername**] *LIBERTY_ADMIN_USERNAME*	The Liberty administrator user name.
[**-lp**|**--libertyadminpassword**] *LIBERTY_ADMIN_PASSWORD*	The Liberty administrator password.
[**-u**|**--mfpfadminusername**] *MFPF_ADMIN_USERNAME*	The MobileFirst Server administrator user name.
[**-pa**|**--mfpfadminpassword**] *MFPF_ADMIN_PASSWORD*	The MobileFirst Server administrator password.

Usage example:

```
removeproject.sh
 --runtime runtime_name
 --ip runtime_ip_addrress
 --port port_number
 --libertyadminusername liberty_administrator
 --libertyadminpassword liberty_password
 --mfpfadminusername mfserver_administrator
 --mfpfadminpassword mfserver_password
```

Table 11-36. Optional command-line arguments

Command-line argument	**Description**
[**-d**|**--deletedata**] *DELETE_RUNTIME_DATA*	Confirmation to delete the project-related data from the database. Accepted values are Y or N (default).
[**-an**|**--appname**] *APP_NAME*	The Bluemix application name.
[**-n**|**--servicename**] *DB_SRV_NAME*	The Bluemix database service instance name.
[**-sn**|**--schema**] *SCHEMA_NAME*	Schema name (When no value is specified, defaults to the runtime nan
[**-ar**|**--adminroot**] *MFPF_ADMIN_ROOT*	Admin context path of the MobileFirst Server (When no value is specified, defaults to *worklightadmin*.)
[**-dp**|**--deleteproject**] *DELETE_RUNTIME_PROJECT*	Confirmation to delete the runtime project from the project folder. Accepted values are Y or N (default).
[**-ce**|**--certpath**] *CERTIFICATE_PATH*	The path to the folder containing server certificates, to perform the HTTPS operations.

Using the MobileFirst Data Proxy:

The MobileFirst Data Proxy can be configured to work with your container so that your mobile app data is stored on the server side.

The MobileFirst Data Proxy works with Cloudant databases. Learn more...

Therefore, to use the MobileFirst Data Proxy with your container, you must have configured a Cloudant database instance to communicate with the MobileFirst Data Proxy component.

For example, this could be a database instance on Cloudant.com or a Cloudant NoSQL DB service on IBM Bluemix.

Using HTTPS

For MobileFirst Data Proxy to use a Cloudant database through HTTPS, you must import the Cloudant database certificate into the truststore of the Java runtime environment (JRE) that is used by the Liberty server. You can find the truststore location in `JRE_DIR/lib/security/cacerts`. Place updated certificate files in the `mfpf-server/usr/jre-security` folder. Learn more...

Using a Cloudant NoSQL DB service on Bluemix

To use the Cloudant NoSQL DB service, it must be bound to a Node.js or IBM Liberty runtime app on Bluemix.

Configuring the MobileFirst Data Proxy to work in a container:

To configure MobileFirst Data Proxy for your container, you can either specify JNDI properties or bind your container to a Bluemix app that has the Cloudant NoSQL DB service.

Before you begin

Before you begin, you must have configured the Cloudant instance that will be used to communicate with the MobileFirst Data Proxy component.

If the Cloudant database is accessed through HTTPS with a self-signed certificate, you must import the certificate to the cacerts truststore of the Java runtime environment (JRE) that is used by the Liberty server. Place updated certificate files in the `mfpf-server/usr/jre-security` folder. Learn more...

Have the following Cloudant account information ready to complete the configuration steps:
- Cloudant proxy database account
- Host name
- User name and password

If you are using the Cloudant NoSQL DB service on Bluemix, make sure that you have already done the following:
- Created an app on Bluemix with a Node.js or Liberty runtime.
- Created an instance of the Cloudant NoSQL DB service.
- Bound the Cloudant NoSQL DB service to the Bluemix app.

Procedure

1. Open `mfpf-server/usr/config/dataproxy.xml` and uncomment the MobileFirst Data Proxy configuration.

2. Specify the public key server URL for the MobileFirst Server project. To do this, open the `mfpf-server/usr/env/server.env` file and then uncomment and set the required `publicKeyServerUrl` environment variable. Example:

   ```
   publicKeyServerUrl=http://mobilefirstserver.mycompany.com:9080/MobileFirstProject
   ```

3. The related Cloudant properties can be specified using JNDI properties in the `mfpf-server/usr/config/dataproxy.xml` file or by binding your container to a Bluemix application that has an instance of the Cloudant NoSQL DB service. Choose one of the following options:

   - To specify JNDI properties, edit the `dataproxy.xml` file according to the following example.

     ```
     <jndiEntry jndiName="datastore/CloudantProxyDbAccount" value='"hostname"'/>
     <jndiEntry jndiName="datastore/CloudantProtocol" value='"http"'/>
     <jndiEntry jndiName="datastore/CloudantPort" value='"80"'/>
     <jndiEntry jndiName="datastore/CloudantProxyDbAccountUser" value='"cloudantuser"'/>
     <jndiEntry jndiName="datastore/CloudantProxyDbAccountPassword" value='"cloudantpassword"'/>
     ```

     Learn more...

   - To bind your container to a Bluemix app that has the Cloudant NoSQL DB service, you must specify the Bluemix app name when running the `startserver.sh` or `startservergroup.sh` script.

     When using this method, the environment variables are obtained automatically.

     Note: Keep in mind that the VCAP_SERVICES environment variables in this process override any environment variables in the container that were specified manually.

MobileFirst Operational Analytics containers

This section contains the information you need to configure and run MobileFirst Operational Analytics containers.

Customizing MobileFirst Operational Analytics containers:

Use the scripts provided in the MobileFirst Platform Foundation on IBM Containers package to customize container images.

The folder where you extracted the contents of the MobileFirst Platform Foundation on IBM Containers package is referred to in this document as the *installation directory*. The customizable elements for the MobileFirst Operational Analytics container are located in `package_root/mfpf-analytics/usr`. See the following tables for details.

Table 11-37. Descriptions of the `mfpf-analytics/` sub directories

Folders and Files	Description
`./usr`	The root folder that contains the customization template for the MobileFirst Operational Analytics container.
`./usr/bin`	Contains the script file (mfp-init) that gets executed when the container starts. You can add custom code to the script, however, do not modify the existing code.
`./usr/config`	Contains the server configuration fragments (keystore, server properties, user registry) used by MobileFirst Server. MobileFirst Operational Analytics container.

Table 11-37. Descriptions of the `mfpf-analytics/` sub directories (continued)

Folders and Files	Description
`./usr/jre-security`	Add JRE security-related files (such as the JRE truststore, policy .jar files, and so forth) to be updated on the container. The files in this folder get copied to the *JAVA_HOME*`/jre/lib/security/` folder in the container.
`./usr/config/keystore.xml`	The configuration of the repository of security certificates used for SSL encryption. The keystore files referenced here should be provided as part of the `./usr/security` folder
`./usr/config/mfpfproperties.xml`	The configuration of the MobileFirst Operational Analytics can be provided here. See the list of supported properties at "Properties and configurations" on page 13-90.
`./usr/config/registry.xml`	User registry configuration. By default provides the basicRegistry - a simple XML based user-registry configuration. The usernames / password for basicRegistry can be configured here. The registry can also be configured for ldapRegistry.
`./usr/env/server.env`	The environment properties used by the server to initialize. Supported properties
./usr/security	The keystore, truststore, and the LTPA keys (ltpa.keys) files should be placed in this folder.
./usr/ssh	The `id_rsa.pub` file - the ssh public key file to enable ssh on the container.

Table 11-38. Server.env supported properties

Property Name	Default Value	Description
ANALYTICS_SERVER_HTTPPORT	9080*	The port used for client HTTP requests. Use -1 to disable this port.
ANALYTICS_SERVER_HTTPSPORT	9443*	The port used for client HTTP requests secured with SSL (HTTPS). Use -1 to disable this port.
ANALYTICS_ADMIN_GROUP	analyticsadmingroup	The name of the user group possessing the predefined role *worklightadmin*.

*Do not modify the default port number. Read more in the following section.

Port number limitation

There is currently an IBM Containers limitation with the port numbers that are available for public domain. Therefore, the default port numbers given for the MobileFirst Operational Analytics container and the MobileFirst Server container (9080 for HTTP and 9443 for HTTPS) cannot be altered. Containers in a container group must use HTTP port 9080. Container groups do not support the use of multiple port numbers or HTTPS requests.

Building and running the MobileFirst Operational Analytics container:

Use the scripts provided in the package installation directory under `mfpf-analytics/scripts` to build and run your customized image.

Before you begin

- Customization of the image has been completed.

About this task

Scripts can only be run from within the `scripts` folder. Do not modify the given folder structure.

The following methods are supported for passing parameters to the scripts:

- Command-line arguments (Usage: `scriptname.sh [-command|--command] ARGUMENT`)
- Interactive method (By running the script with no command-line arguments.)
- Properties files (By customizing the related `args/*.properties` files.)

  You can find information about the required and optional arguments in the properties files, such as the default values, input descriptions, and so forth. The script properties files are located in the following folders in the *package_root/*:

  mfpf-analytics`/scripts/args`

  mfpf-server`/scripts/args`

  Example command execution usage: `prepareserver.sh args/prepareserver.properties`

Procedure

In the following procedure, the first step describes how to retrieve a public IP address to bind to the container, which is a required argument when you run the `startanalytics.sh` script (as described in step 2).

1. Retrieve and take note of a public IP address to bind to the container.

   To get IP information, use Cloud Foundry CLI plug-in for IBM Containers (**`cf ic`**) commands.

   - To retrieve the list of IP addresses that are potentially available to you (based on user ID), run **`cf ic ip list`**.

     The IP addresses that are listed with no corresponding container ID are available for use.

     An IP address with a corresponding container ID indicates that the IP address is already in use. If all IP addresses are already in use, you can request a new IP address.

   - To request a new IP address, run **`cf ic ip request`**.

2. Run the following scripts in order:

   `initenv.sh`
   This script logs in to the container service. You must run this script before you can run any subsequent scripts.

   `prepareanalytics.sh`
   This script builds the analytics image with your customizations and deploys it to IBM Containers.

   `startanalytics.sh`
   The script runs the analytics image as a standalone container.

   `startanalyticsgroup.sh`
   The script runs the analytics image as a container group.

Script overview and usage:

Use the scripts for configuring, building, and deploying a MobileFirst Operational Analytics container image.

The scripts are located in the *package_root*/mfpf-analytics/scripts folder.

initenv.sh script:

This script file creates the environment for building and running a MobileFirst Operational Analytics container and performs the tasks necessary for logging into Bluemix and the IBM Containers environment. You must run this script before running any other scripts for building and deploying MobileFirst Operational Analytics container images.

Table 11-39. Mandatory command-line arguments

Command-line argument	Description
[**-u**|**--user**] *BLUEMIX_USER*	Bluemix user ID or email address
[**-p**|**--password**] *BLUEMIX_PASSWORD*	Bluemix password
[**-o**|**--org**] *BLUEMIX_ORG*	Bluemix organization name
[**-s**|**--space**] *BLUEMIX_SPACE*	Bluemix space name

Usage example:

```
initenv.sh
  --user Bluemix_user_ID
  --password Bluemix_password
  --org Bluemix_organization_name
  --space Bluemix_space_name
```

Table 11-40. Optional command-line arguments

Command-line argument	Description
[**-a**|**--api**] *BLUEMIX_API_URL*	Bluemix API endpoint (Defaults to https://api.ng.bluemix.net)

prepareanalytics.sh script:

This script creates the MobileFirst Operational Analytics image and pushes it to IBM Containers on Bluemix. Ensure that you have run the initenv.sh before running this script.

Table 11-41. Mandatory command-line arguments

Command-line argument	Description
[**-t**|**--tag**] *ANALYTICS_IMAGE_TAG*	Name to be used for the customized analytics image. Format: *Bluemix registry URL/private namespace/image name*

Usage example:

```
prepareanalytics.sh --tag registry.ng.bluemix.net/your_private_repository_namespace/mfpfanalytics71
```

startanalytics.sh script:

This script runs the MobileFirst Operational Analytics image as a container on IBM Containers on Bluemix. Before running this script, ensure that you have run the `prepareanalytics.sh` script to upload the image to the IBM Containers registry.

Table 11-42. Mandatory command-line arguments

Command-line argument	Description
[**-t**|**--tag**] *ANALYTICS_IMAGE_TAG*	Name of the analytics container image that has been loaded into the IBM Containers registry. Format: *`BluemixRegistry`*`/`*`PrivateNamespace`*`/`*`ImageName`*`:`*`Tag`*
[**-n**|**--name**] *ANALYTICS_CONTAINER_NAME*	Name of the analytics container
[**-i**|**--ip**] *ANALYTICS_IP*	IP address that the container should be bound to. (You can provide an available public IP or request one using the **cf ic ip request** command.)

Usage example:

```
startanalytics.sh
 --tag image_tag_name
 --name container_name
 --ip container_ip_address
```

Table 11-43. Optional command-line arguments

Command-line argument	Description
[**-h**|**--http**] *EXPOSE_HTTP*	Expose HTTP port. Accepted values are Y (default) or N.
[**-s**|**--https**] *EXPOSE_HTTPS*	Expose HTTPS port. Accepted values are Y (default) or N.
[**-m**|**--memory**] *SERVER_MEM*	Assign a memory size limit to the container in megabytes (MB). Accepted values are 1024 MB (default) and 2048 MB.
[**-se**|**--ssh**] *SSH_ENABLE*	Enable SSH for the container. Accepted values are Y (default) or N.
[**-sk**|**--sshkey**] *SSH_KEY*	The SSH Key to be injected into the container. (Provide the contents of your `id_rsa.pub` file.)
[**-tr**|**--trace**] *TRACE_SPEC*	The trace specification to be applied. Default: **=info*
[**-ml**|**--maxlog**] *MAX_LOG_FILES*	The maximum number of log files to maintain before they are overwritten. The default is 5 files.
[**-ms**|**--maxlogsize**] *MAX_LOG_FILE_SIZE*	The maximum size of a log file. The default size is 20 MB.
[**-v**|**--volume**] *ENABLE_VOLUME*	Enable mounting volume for container logs. Accepted values are Y or N (default).
[**-ev**|**--enabledatavolume**] *ENABLE_ANALYTICS_DATA_VOLUME*	Enable mounting volume for analytics data. Accepted values are Y or N (default).

Table 11-43. Optional command-line arguments (continued)

Command-line argument	Description
[**-av**|**--datavolumename**] *ANALYTICS_DATA_VOLUME_NAME*	Specify the name of the volume to be created and mounted for the analytic data. The default name is `mfpf_analytics_`*`container_name`*.
[**-ad**|**--analyticsdatadirectory**] *ANALYTICS_DATA_DIRECTORY*	Specify the location to store the data. The default folder name is */analyticsData*.
[**-e**|**--env**] **MFPF_PROPERTIES**	Provide MobileFirst Operational Analytics properties as comma-separated key:value pairs. Note: If you specify properties using this script, ensure tha

startanalyticsgroup.sh script:

This script runs a MobileFirst Operational Analytics image as a container group on IBM Containers on Bluemix. Before running `startanalyticsgroup.sh`, ensure that you have run the `prepareanalytics.sh` script to upload the container image to the IBM Containers registry.

Table 11-44. Mandatory command-line arguments

Command-line argument	Description
[**-t**|**--tag**] *ANALYTICS_IMAGE_TAG*	The name of the analytics container image in the Bluemix registry.
[**-gn**|**--name**] *ANALYTICS_CONTAINER_GROUP_NAME*	The name of the analytics container group.
[**-gh**|**--host**] *ANALYTICS_CONTAINER_GROUP_HOST*	The host name of the route.
[**-gs**|**--domain**] *ANALYTICS_CONTAINER_GROUP_DOMAIN*	The domain name of the route.

Usage example:

```
startanalyticsgroup.sh
 --tag image_name
 --name container_group_name
 --host container_group_host_name
 --domain container_group_domain_name
```

Table 11-45. Optional command-line arguments

Command-line argument	Description
[**-gm**|**--min**] **ANALYTICS_CONTAINER_GROUP_MIN**	The minimum number of container instances. The default value is 1.
[**-gx**|**--max**] *ANALYTICS_CONTAINER_GROUP_MAX*	The maximum number of container instances. The default value is 1.
[**-gd**|**--desired**] *ANALYTICS_CONTAINER_GROUP_DESIRED*	The desired number of container instances. The default value is 2.
[**-a**|**--auto**] *ENABLE_AUTORECOVERY*	Enable the automatic recovery option for the container instances. Accepted values are Y or N (default).

Table 11-45. Optional command-line arguments (continued)

Command-line argument	Description
[**-tr**|**--trace**] *TRACE_SPEC*	The trace specification to be applied. Default: **=info*
[**-ml**|**--maxlog**] *MAX_LOG_FILES*	The maximum number of log files to maintain before they are overwritten. The default is 5 files.
[**-ms**|**--maxlogsize**] *MAX_LOG_FILE_SIZE*	The maximum size of a log file. The default size is 20 MB.
[**-e**|**--env**] *MFPF_PROPERTIES*	Specify MobileFirst properties as comma-separated key:value pairs. Example: *wl.analytics.url:http://127.0.0.1/analytics-service/v2*
[**-m**|**--memory**] *SERVER_MEM*	Assign a memory size limit to the container in megabytes (MB). Accepted values are 1024 MB (default) and 2048 MB.
[**-v**|**--volume**] *ENABLE_VOLUME*	Enable mounting volume for container logs. Accepted values are Y or N (default).
[**-ev**|**--enabledatavolume**] *ENABLE_ANALYTICS_DATA_VOLUME*	Enable mounting volume for analytics data. Accepted values are Y or N (default)
[**-av**|**--datavolumename**] *ANALYTICS_DATA_VOLUME_NAME*	Specify name of the volume to be created and mounted for analytics data. Default value is mfpf_analytics_<ANALYTICS_CONTAINER_GROUP_NAME>
[**-ad**|**--analyticsdatadirectory**] *ANALYTICS_DATA_DIRECTORY*	Specify the directory to be used for storing analytics data. Default value is /analyticsData"

Learn more about creating scalable container groups.

Securing containers

Security configuration for IBM MobileFirst Platform Foundation for iOS on IBM Containers:

Your IBM MobileFirst Platform Foundation for iOS on IBM Containers security configuration should include encrypting passwords, enabling application authenticity checking, and securing access to the consoles.

Encrypting passwords

Store the passwords for MobileFirst Server users in an encrypted format. You can use the **securityUtility** command available in the Liberty profile to encode passwords with either XOR or AES encryption. Encrypted passwords can then be copied into the /usr/env/server.env file. See "Encrypting passwords for user roles configured in MobileFirst Server" on page 11-132 for instructions.

Application authenticity checking

To keep unauthorized mobile applications from accessing the MobileFirst Server, enable application authenticity. Learn more...

Configure SSL for Operations Console and Analytics Console

You can secure access to the MobileFirst Operations Console and the MobileFirst Analytics Console by enabling HTTP over SSL (HTTPS) on the MobileFirst Server.

To enable HTTPS on the MobileFirst Server, create the keystore containing the certificate and place it in the `usr/security` folder. Then, update the `usr/config/keystore.xml` file to use the keystore configured.

Securing a connection to the back end

If you need a secure connection between your container and an on-premise back-end system, you can use the Bluemix Secure Gateway service. Configuration details are provided in this article: Connecting Securely to On-Premise Backends from MobileFirst on IBM Bluemix containers.

Encrypting passwords for user roles configured in MobileFirst Server:

The passwords for user roles that are configured for the MobileFirst Server can be encrypted.

Procedure

Passwords are configured in the `server.env` files in the *package_root*`/mfpf-server/usr/env` and *package_root*`/mfpf-analytics/usr/env` folders. Passwords should be stored in an encrypted format.

1. You can use the securityUtility command in the Liberty profile to encode the password. Choose either XOR or AES encryption to encode the password.
2. Copy the encrypted password to the `server.env` file. Example: `MFPF_ADMIN_PASSWORD={`*xor*`}`*PjsyNjE=*
3. If you are using AES encryption and used your own encryption key instead of the default key, you must create a configuration file that contains your encryption key and add it to the `usr/config` directory. The Liberty server accesses the file to decrypt the password during runtime. The configuration file must have the `.xml` file extension and resemble the following format:

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
  <variable name="wlp.password.encryption.key" value="yourKey" />
</server>
```

Securing container communication using a private IP address:

To have secure communication between the MobileFirst Server container and the MobileFirst Operational Analytics container, you must include the private IP address of the MobileFirst Operational Analytics container in the `mfpfProperties.xml` file.

Before you begin

To complete this task, you need the private IP of the MobileFirst Operational Analytics container, which you can obtain using the following command: `cf ic inspect` *analytics_container_id*. Look for the IP Address field in the command output.

Remember: If you are going to use MobileFirst Operational Analytics, you must configure, build, and run the MobileFirst Operational Analytics image before configuring, deploying, and running the MobileFirst Server image.

Procedure

Complete the following steps by editing the `mfpf-server/usr/config/mfpfproperties.xml` file:

1. Set the `wl.analytics.url` property to the private IP address of the MobileFirst Operational Analytics container. Example:

   ```
   <jndiEntry jndiName="wl.analytics.url" value="http://AnalyticsContainerPrivateIP:9080/analytic
   ```

   Tip: When a private IP address changes, provide the new IP address in the `mfpfproperties.xml` file and rebuild and deploy the container by running the `prepareserver.sh` and `starterserver.sh` scripts respectively.
2. To ensure that the MobileFirst Operational Analytics console can be accessed on the network, set the `wl.analytics.console.url` property to the public IP address of the MobileFirst Operational Analytics container. Example:

   ```
   <jndiEntry jndiName="wl.analytics.console.url" value="http://AnalyticsContainerPublicIP:9080/a
   ```

Restricting access to the consoles running on containers:

You can restrict access to the MobileFirst Operations Console and the MobileFirst Analytics Console in production environments by creating and deploying a Trust Association Interceptor (TAI) to intercept requests to the consoles running on IBM Containers.

About this task

The TAI can implement user-specific filtering logic that decides if a request is forwarded to the console or if an approval is required. This method of filtering provides the flexibility for you to add your own authentication mechanism if needed.

See also: Developing a custom TAI for the Liberty profile

Procedure

1. Create a custom TAI that implements your security mechanism to control access to the MobileFirst Operations Console. The following example of a custom TAI uses the IP Address of the incoming request to validate whether to provide access to the MobileFirst Operations Console or not.

   ```
   package com.ibm.mfpconsole.interceptor;
   import java.util.Properties;

   import javax.servlet.http.HttpServletRequest;
   import javax.servlet.http.HttpServletResponse;

   import com.ibm.websphere.security.WebTrustAssociationException;
   import com.ibm.websphere.security.WebTrustAssociationFailedException;
   import com.ibm.wsspi.security.tai.TAIResult;
   import com.ibm.wsspi.security.tai.TrustAssociationInterceptor;

   public class MFPConsoleTAI implements TrustAssociationInterceptor {

      String allowedIP =null;

      public MFPConsoleTAI() {
   ```

```
      super();
   }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#isTargetInterceptor
 * (javax.servlet.http.HttpServletRequest)
 */
   public boolean isTargetInterceptor(HttpServletRequest req)
                  throws WebTrustAssociationException {
      //Add logic to determine whether to intercept this request

    boolean interceptMFPConsoleRequest = false;
    String requestURI = req.getRequestURI();

    if(requestURI.contains("worklightConsole")) {
     interceptMFPConsoleRequest = true;
    }

    return interceptMFPConsoleRequest;
   }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#negotiateValidateandEstablishTrust
 * (javax.servlet.http.HttpServletRequest,javax.servlet.http.HttpServletResponse)
 */
   public TAIResult negotiateValidateandEstablishTrust(HttpServletRequest request,
                    HttpServletResponse resp) throws WebTrustAssociationFailedException {
        // Add logic to authenticate a request and return a TAI result.
        String tai_user = "MFPConsoleCheck";

        if(allowedIP != null) {

         String ipAddress = request.getHeader("X-FORWARDED-FOR");
         if (ipAddress == null) {
           ipAddress = request.getRemoteAddr();
         }

         if(checkIPMatch(ipAddress, allowedIP)) {
          TAIResult.create(HttpServletResponse.SC_OK, tai_user);
         }
         else {
          TAIResult.create(HttpServletResponse.SC_FORBIDDEN, tai_user);
         }

        }
        return TAIResult.create(HttpServletResponse.SC_OK, tai_user);
    }

   private static boolean checkIPMatch(String ipAddress, String pattern) {

    if (pattern.equals("*.*.*.*") || pattern.equals("*"))
        return true;

    String[] mask = pattern.split("\\.");
    String[] ip_address = ipAddress.split("\\.");

    for (int i = 0; i < mask.length; i++)
    {
     if (mask[i].equals("*") || mask[i].equals(ip_address[i]))
        continue;
     else
        return false;
  }
  return true;
   }

/*
```

```
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#initialize(java.util.Properties
 */
    public int initialize(Properties properties)
                    throws WebTrustAssociationFailedException {

     if(properties != null) {
      if(properties.containsKey("allowedIPs")) {
       allowedIP = properties.getProperty("allowedIPs");
      }
     }
        return 0;
    }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getVersion()
 */
    public String getVersion() {
        return "1.0";
    }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#getType()
 */
    public String getType() {
        return this.getClass().getName();
    }

/*
 * @see com.ibm.wsspi.security.tai.TrustAssociationInterceptor#cleanup()
 */
    public void cleanup()

    {}
}
```

2. Export the custom TAI Implementation into a .jar file and place it in the applicable env folder (mfpf-server/usr/env or mfpf-analytics/usr/env).
3. Create an XML configuration file that contains the details of the TAI interceptor (see the TAI configuration example code provided in step 1) and then add your .xml file to the applicable folder (mfpf-server/usr/config or mfpf-analytics/usr/config). Your .xml file should resemble the following example.

   Tip: Be sure to update the class name and properties to reflect your implementation.

```
<?xml version="1.0" encoding="UTF-8"?>
<server description="new server">
<featureManager>
    <feature>appSecurity-2.0</feature>
</featureManager>

<trustAssociation id="MFPConsoleTAI" invokeForUnprotectedURI="true"
                  failOverToAppAuthType="false">
    <interceptors id="MFPConsoleTAI" enabled="true"
                  className="com.ibm.mfpconsole.interceptor.MFPConsoleTAI"
                  invokeBeforeSSO="true" invokeAfterSSO="false" libraryRef="MFPConsoleTAI">
        <properties allowedIPs="9.182.149.*"/>
    </interceptors>
</trustAssociation>

<library id="MFPConsoleTAI">
    <fileset dir="${server.config.dir}" includes="MFPConsoleTAI.jar"/>
</library>

</server>
```

4. Build the image and run the container as described in “Building and running the MobileFirst Server container” on page 11-114 or “Building and running the MobileFirst Operational Analytics container” on page 11-126. The MobileFirst Operations Console and the Analytics Console are now accessible only when the configured TAI security mechanism is satisfied.

Configuring App Transport Security (ATS):

This topic outlines how to configure App Transport Security (ATS) for MobileFirst Server and MobileFirst Operational Analytics containers.

Before you begin

Before you begin, review Requirements for Connecting Using ATS.

About this task

ATS configuration does not impact applications connecting from other, non-iOS, mobile operating systems. Other mobile operating systems do not mandate that servers communicate on the ATS level of security but can still communicate with ATS-configured servers.

Before configuring your container image, have the generated certificates ready. The following steps assume that the keystore file `*ssl_cert*.p12` has the personal certificate and `ca.crt` is the signing certificate.

Procedure

1. Copy the `*ssl_cert*.p12` file to the `*mfpf-server*/usr/security/` folder.
2. Modify the `*mfpf-server*/usr/config/keystore.xml` file similar to the following example configuration:

```
<server>
   <featureManager>
      <feature>ssl-1.0</feature>
   </featureManager>
   <ssl id="defaultSSLConfig" sslProtocol="TLSv1.2" keyStoreRef="defaultKeyStore" enabledCiphers=
   <keyStore id="defaultKeyStore" location="ssl_cert.p12" password="*****" type="PKCS12"/>
</server>
```

   - *ssl-1.0* is added as a feature in the feature manager to enable the server to work with SSL communication.
   - *sslProtocol="TLSv1.2"* is added in the `ssl` tag to mandate that the server communicates only on Transport Layer Security (TLS) version 1.2 protocol.

     More than one protocol can be added. For example, adding `sslProtocol=`*"TLSv1+TLSv1.1+TLSv1.2"* would ensure that the server could communicate on TLS V1, V1.1, and V1.2. (TLS V1.2 is required for iOS 9 apps.)
   - *enabledCiphers="TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384"* is added in the `ssl` tag so that the server enforces communication using only that cipher.

     More ciphers can be added to the list. The server will communicate with the accepted iOS 9 ciphers that have been added to this list.
   - The `keyStore` tag tells the server to use the new certificates that are created as per the above requirements.

What to do next

The following specific ciphers require Java Cryptography Extension (JCE) policy settings and an additional JVM option:

TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384

TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384

If you use these ciphers and use an IBM Java SDK, you can download the policy files. There are two files: `US_export_policy.jar` and `local_policy.jar`. Add both the files to the `mfpf-server/usr/security` folder and then add the following JVM option to the `mfpf-server/usr/env/jvm.options` file: `Dcom.ibm.security.jurisdictionPolicyDir=/opt/ibm/wlp/usr/servers/worklight/resources/security/`.

For development-stage purposes only, you can disable ATS by adding following property to the `info.plist` file:

```
<key>NSAppTransportSecurity</key>
  <dict>
   <key>NSAllowsArbitraryLoads</key>
   <true/>
  </dict>
```

See also ATS and Bitcode in iOS 9.

LDAP configuration for containers:

You can configure an IBM MobileFirst Platform Foundation for iOS container to securely connect out to an external LDAP repository.

The external LDAP registry can be used in a container for the following purposes:
- To configure the MobileFirst administration security with an external LDAP registry.
- To configure the MobileFirst mobile applications to work with an external LDAP registry.

Configuring administration security with LDAP:

Configure the MobileFirst administration security with an external LDAP registry

About this task

The configuration process includes the following steps:
- Set up and configuration of an LDAP repository
- Changes to the registry file (registry.xml)
- Configuration of a secure gateway to connect to a local LDAP repository and the container. (You need an existing app on Bluemix for this step.)

Procedure

LDAP repository

1. Create users and groups in the LDAP repository. For groups, authorization is enforced based on user membership.

Registry file

2. Open registry.xml and find the basicRegistry element. Replace the basicRegistry element with code that is similar to the following snippet:

   ```
   <ldapRegistry id="ldap"
   host="1.234.567.8910" port="1234" ignoreCase="true"
   baseDN="dc=worklight,dc=com"
   ldapType="Custom"
   sslEnabled="false"
   bindDN="uid=admin,ou=system"
   bindPassword="secret">
   <customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
   groupFilter="(&amp;(member=uid=%v)(objectclass=groupOfNames))"
   userIdMap="*:uid"
   groupIdMap="*:cn"
   groupMemberIdMap="groupOfNames:member"/>
   </ldapRegistry>
   ```

Table 11-46. Descriptions of the ldapRegistry entries

Entry	Description
host and port	Host name (IP address) and port number of your local LDAP server.
baseDN	The domain name (DN) in LDAP that captures all details about a specific organization.
bindDN="uid=*admin*,ou=*system*"	Binding details of the LDAP server. For example, the default values for an Apache Directory Service would be uid=admin,ou=system.
bindPassword="*secret*"	Binding password for the LDAP server. For example, the default value for an Apache Directory Service is secret.

<customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
groupFilter="(&amp;(member=uid=%v)(objectclass=groupOfNames))"
userIdMap="*:uid"
groupIdMap="*:cn"
groupMemberIdMap="groupOfNames:member"/>
	The custom filters that are used for querying the directory service (such as Apache) during authentication and authorization.

3. Ensure that the following features are enabled for appSecurity-2.0 and ldapRegistry-3.0:

   ```
   <featureManager>
   <feature>appSecurity-2.0</feature>
   <feature>ldapRegistry-3.0</feature>
   </featureManager>
   ```

   For details about configuring various LDAP server repositories, see the WebSphere Application Server Liberty Knowledge Center.

After you complete the registry.xml changes, configure a secure gateway to connect to the local LDAP server.

Secure gateway

To configure a secure gateway connection to your LDAP server, you must create an instance of the Secure Gateway service on Bluemix and then obtain the IP information for the LDAP registry. You need your local LDAP host name and port number for this task.

4. Log on to Bluemix and navigate to **Catalog**, **Category** > **Integration**, and then click **Secure Gateway**.
5. Under Add Service, select an app and then click **CREATE**. Now the service is bound to your app.
6. Go to the Bluemix dashboard for the app, click on the **Secure Gateway** service instance, and then click **ADD GATEWAY**.
7. Name the gateway and click **ADD DESTINATIONS** and enter the name, IP address, and port for your local LDAP server.
8. Follow the prompts to complete the connection. To see the destination initialized, navigate to the Destination screen of the LDAP gateway service.
9. To obtain the host and port information that you need, click the Information icon on the LDAP gateway service instance (located on the Secure Gateway dashboard). The details displayed are an alias to your local LDAP server.
10. Capture the **Destination ID** and **Cloud Host : Port** values. Go to the registry.xml file and add these values, replacing any existing values. See the following example of an updated code snippet in the registry.xml file:

```
<ldapRegistry id="ldap"
host="cap-sg-prd-5.integration.ibmcloud.com" port="15163" ignoreCase="true"
baseDN="dc=worklight,dc=com"
ldapType="Custom"
sslEnabled="false"
bindDN="uid=admin,ou=system"
bindPassword="secret">
<customFilters userFilter="(&amp;(uid=%v)(objectclass=inetOrgPerson))"
groupFilter="(&amp;(member=uid=%v)(objectclass=groupOfNames))"
userIdMap="*:uid"
groupIdMap="*:cn"
groupMemberIdMap="groupOfNames:member"/>
</ldapRegistry>
```

Configuring apps to work with LDAP:

Configure MobileFirst mobile apps to work with an external LDAP registry

About this task

The configuration process includes the following steps:

- Configuring the LDAP login module in the MobileFirst project.
- Configuring a secure gateway to connect to a local LDAP repository and the container. (You need an existing app on Bluemix for this step.)

Procedure

1. Configure the LDAP login module in your MobileFirst project to authenticate the mobile app users against the LDAP server.

To configure a secure gateway connection to your LDAP server, you must create an instance of the Secure Gateway service on Bluemix and then obtain the IP information for the LDAP registry. You need your local LDAP host name and port number for this step.

2. Log on to Bluemix and navigate to **Catalog**, **Category** > **Integration**, and then click **Secure Gateway**.
3. Under Add Service, select an app and then click **CREATE**. Now the service is bound to your app.
4. Go to the Bluemix dashboard for the app, click on the **Secure Gateway** service instance, and then click **ADD GATEWAY**.
5. Name the gateway and click **ADD DESTINATIONS** and enter the name, IP address, and port for your local LDAP server.
6. Follow the prompts to complete the connection. To see the destination initialized, navigate to the Destination screen of the LDAP gateway service.
7. To obtain the host and port information that you need, click the Information icon on the LDAP gateway service instance (located on the Secure Gateway dashboard). The details displayed are an alias to your local LDAP server.
8. Capture the **Destination ID** and **Cloud Host : Port** values. Provide these values for the LDAP login module.

Results

The communication between the MobileFirst app in the container on Bluemix with your local LDAP server is established. The authentication and authorization from the Bluemix app is validated against your local LDAP server.

Removing a container from Bluemix

When you remove a container from Bluemix, you must also remove the image name from the registry.

Procedure

1. Run the following Cloud Foundry CLI plug-in for IBM Containers (**`cf ic`**) commands to remove a container from Bluemix:
   a. `cf ic ps` (Lists the containers currently running)
   b. `cf ic stop` *`container_id`* (Stops the container)
   c. `cf ic rm` *`container_id`* (Removes the container)
2. Run the following **`cf ic`** commands to remove an image name from the Bluemix registry:
   a. `cf ic images` (Lists the images in the registry)
   b. `cf ic rmi` *`image_id`* (Removes the image from the registry)

Removing the database service configuration app from Bluemix

If you ran the **`prepareserverdbs.sh`** script during the configuration of your MobileFirst Platform Foundation on IBM Containers image, a Bluemix app with a bound database service was created that ultimately created the database schema for the container.

About this task

To remove the database service configuration app from Bluemix, you can either use your Bluemix dashboard or run the following Cloud Foundry commands:

```
cf delete app_name
cf delete-service service_name
```

Log and trace collection

Syslog and Liberty log files are generated for MobileFirst Server and MobileFirst Operational Analytics containers.

To persist the log files, you must enable volume. Syslogs are persisted in the `/var/log/rsyslog` folder and Liberty logs are persisted in the `/opt/ibm/wlp/usr/servers/worklight/logs` folder.

Liberty logs get created at the trace specification level *info* by default (collecting general information that outlines task progress), however, you can change the logging level by manually adding a code override or by injecting code using a script file.

Enabling SSH to access container logs, enabling volume to persist log files, changing the trace level, and other logging-related options must be done during container creation when running one of the following start-up scripts:

“startserver.sh script” on page 11-120

“startservergroup.sh script” on page 11-121

“startanalytics.sh script” on page 11-129

“startanalyticsgroup.sh script” on page 11-130

Volume

When you enable the volume configuration option for a container, volumes persist the container logs regardless of whether the container is active or deleted. Volume for MobileFirst Server and MobileFirst Operational Analytics containers is not enabled by default.

You can enable volume when running one of the start-up scripts mentioned above.

Examples of how to enable volume using a start-up script for regular, silent, and interactive command modes:

- Command-line mode - If you are running scripts from the command line, pass-in the value `Y` for the [**-v**|**--volume**] command-line argument.
- Silent mode - If you are running scripts in silent mode, set the ENABLE_VOLUME property to Y in the related properties file for the script (such as `args/startserver.properties` or `args/startservergroup.properties`).
- Interactive mode - If you are running scripts in interactive mode, specify `Y` at the `Enable mounting volume for the MobileFirst container logs` prompt.

You can still view logs in the Bluemix dashboard using the Logmet service regardless of whether volume and SSH were enabled during container creation.

Monitoring

IBM Containers for Bluemix provides some built-in logging and monitoring capabilites around container CPU, memory, and the interfacing networks. Both container logs and server or runtime logs can be viewed from a Bluemix logmet console by means of the Kibana visualization tool. Active monitoring can be done from a Bluemix logmet console with Grafana, an open source metrics dashboard and graph editor.

See also:

Monitoring and logging for IBM Containers

IBM Containers troubleshooting

Accessing log files:

Logs are created for each container instance. You can access log files using the IBM Containers plug-in (**`cf ic`**) commands, the Bluemix™ logmet console, and the IBM Containers REST API. Alternatively, you can obtain container log files through Secure Shell (SSH).

Using SSH with containers:

When your IBM MobileFirst Platform Foundation for iOS container is created with a Secure Shell (SSH) key and bound to a public IP address, a private key can be used to securely view and download the logs for each container instance.

Before you begin

Prerequisites for accessing your MobileFirst Server and MobileFirst Operational Analytics containers to get syslog and Liberty logs:

- SSH must be enabled for the container.
- Volume has been enabled so that the log files are persisted.

For container groups, a public IP address must be bound to each container instance to view the logs using SSH.

Procedure

1. To enable SSH, copy the SSH public key to the *`package_root`*`/[`*`mfpf-server`* `or` *`mfpf-analytics`*`]/usr/ssh` folder before you run the `prepareserver.sh` or the `prepareanalytics.sh` scripts. This builds the image with SSH enabled. Any container created from that particular image will have the SSH enabled.

   If SSH is not enabled as part of the image customization, you can enable it for the container using the SSH_ENABLE and SSH_KEY arguments when executing the start-up scripts. You can optionally customize the related script `.properties` files to include the key content.
2. Make an SSH request to the container. Example:
   ```
   mylocal-workstation# ssh -i
   ~/ssh_key_directory/id_rsa
   root@public_ip
   ```
3. Archive the log file location. Example:
   ```
   container_instance@root# cd
   /opt/ibm/wlp/usr/servers/worklight
   ```
   ```
   container_instance@root# tar czf
   logs_archived.tar.gz logs/
   ```
4. Download the log archive to your local workstation. Example:
   ```
   mylocal-workstation# scp -i
   ~/ssh_key_directory/id_rsa
   root@public_ip:/opt/ibm/wlp/usr/servers/worklight/logs_archived.tar.gz
                                  /local_workstation_dir/target_location/
   ```

Accessing containers from the command line:

You can access running MobileFirst Server and MobileFirst Operational Analytics container instances from the command line to obtain logs and traces.

Before you begin

- The container instance must be in a running state.

Procedure

1. Create an interactive terminal within the container instance by running the following command: `cf ic exec -it` *container_instance_id* `"bash"`.
2. To locate the log files or traces, use the following command example:

   ```
   container_instance@root# cd /opt/ibm/wlp/usr/servers/worklight
   container_instance@root# vi messages.log
   ```
3. To copy the logs to your local workstation, use the following command example:

   ```
   my_local_workstation# cf ic exec -it container_instance_id
    "cat" " /opt/ibm/wlp/usr/servers/worklight/messages.log" > /tmp/local_messages.log
   ```

Using IBM Containers REST API:

The container logs endpoint gets `stdout` logs with the given ID of the container instance.

Before you begin

Example: `GET /containers/`*container_id*`/logs`

Each API request requires an HTTP header X-Auth-Token set with a token obtained with the request authorization token API.

```
X-Auth-Token = Bluemix JWT token
X-Auth-Project-Id = Space GUID
```

See also:

Getting Container logs using IBM Container Cloud Service REST API

Logging overrides:

You can change the log levels by either adding a code override manually or by injecting code using a given script file.

Adding a code override manually to change the log level must be done when you are first preparing the image. You must add the new logging configuration to the *package_root*`/mfpf-[`*analytics*`|`*server*`]/usr/config` folder as a separate configuration snippet, which gets copied to the `configDropins/overrides` folder on the Liberty server.

Injecting code using a given script file to change the log level can be accomplished by using certain command-line arguments when running any of the `start*.sh` script files provided in the MobileFirst Platform Foundation on IBM Containers package (`startserver.sh`, `startanalytics.sh`, `startservergroup.sh`, `startanalyticsgroup.sh`). The following optional command-line arguments are applicable:

`[-tr|--trace]` *trace_specification*

`[-ml||--maxlog]` *maximum_number_of_log_files*

`[-ms|--maxlogsize]` *maximum_size_of_log_files*

Troubleshooting tips

Here are tips for resolving common problems that might occur.

Application not configured correctly:

The following error occurs while running the prepareserverdbs.sh script: `Error : Application not configured correctly.`

Explanation

When you run the prepareserverdbs.sh script, a series of configuration steps take place:

- A Bluemix app is created and started
- A database service is created and bound to the app
- A REST service is invoked in the app and the database schema is created

If there is a delay with the Bluemix app starting, the client is not able to invoke the reset interface, which could cause the error. Or, there could be an error related to the binding.

How to resolve

Run the same script again. If the same error occurs, verify that the Bluemix app is bound to the database service. If not bound, manually bind the database service to the app and then run the script again.

Docker-related error while running script:

If you encounter Docker-related errors after executing the `initenv.sh` or `prepareserver.sh` scripts, try restarting the Docker service.

Example message:

```
* Pulling repository docker.io/library/ubuntu
* Error while pulling image: Get https://index.docker.io/v1/repositories/library/ubuntu/images: dial
```

Explanation

The error could occur when the internet connection has changed (such as connecting to or disconnecting from a VPN or network configuration changes) and the Docker runtime environment has not yet restarted. In this scenario, errors would occur when any Docker command is issued.

How to resolve

Restart the Docker service. If the error persists, reboot the computer and then restart the Docker service.

Bluemix registry error:

If you encounter a registry-related error after executing the `prepareserver.sh` or `prepareanalytics.sh` scripts, try running the `initenv.sh` script first.

Explanation

In general, any network problems that occur while the `prepareserver.sh` or `prepareanalytics.sh` scripts are running could cause processing to hang and then fail.

How to resolve

First, run the `initenv.sh` script again to log in to the container registry on Bluemix. Then, rerun the script that previously failed.

Unable to create the runtime .xml file:

An error occurs at the end of running the prepareserverdbs.sh script: `Error : unable to create` *`runtime_name`*`.xml`

How to resolve

If the .war artifacts for the runtime are in place within the projects folder, then the problem might be an intermittent database connectivity issue. Try to run the script again.

Taking a long time to push image:

When running the prepareserver.sh script, it takes more than 20 minutes to push an image to the IBM Containers Extension (ICE) registry.

Explanation

The prepareserver.sh script pushes the entire IBM MobileFirst Platform Foundation for iOS stack, which can take from 20 to 60 minutes.

How to resolve

If the script has not completed after a 60-minute time period has elapsed, the process might be hung because of a connectivity issue. After a stable connection is reestablished, restart the script.

Binding is incomplete error:

When running a script to start a container (such as startserver.sh or startanalytics.sh) you are prompted to manually bind an IP address because of an error that the binding is incomplete.

Explanation

The script is designed to exit after a certain time duration has passed.

How to resolve

Manually bind the IP address by running the related **`cf ic`** command. For example, `cf ic ip bind`.

If binding the IP address manually is not successful, ensure that the status of the container is running and then try binding again.

Note: Containers must be in a running state to be bound successfully.

Script fails and returns message about tokens:

Running a script is not successful and returns a message similar to `Refreshing cf tokens` or `Failed to refresh token`.

Explanation

The Bluemix session might have timed-out. The user must be logged in to Bluemix before running the container scripts.

How to resolve

Run the initenv.sh script again to log in to Bluemix and then run the failed script again.

No runtimes listed in console:

There are no runtimes listed in the IBM MobileFirst Platform Operations Console even though the prepareserver.sh script completed successfully.

Explanation

It is possible that a either a connection to the database service did not get established or that a formatting problem occurred in the server.env file when additional values were appended during deployment.

If additional values were added to the server.env file without new line characters, the properties would not resolve. You can validate this potential problem by checking the log files for errors caused by unresolved properties that look similar to this error:

```
FWLSE0320E: Failed to check whether the admin services are ready. Caused by: [project Sample] java.n
```

How to resolve

Manually restart the containers. If the problem still exists, check to see if the number of connections to the database service exceeds the number of connections provisioned by your database plan. If so, make any needed adjustments before proceeding.

If the problem was caused by unresolved properties, ensure that your editor adds the linefeed (LF) character to demarcate the end of a line when editing any of the provided files. For example, the TextEdit app on OS X might use the CR character to mark the end of line instead of LF, which would cause the issue.

prepareserver.sh script fails:

The `prepareserver.sh` script fails and returns the error `405 Method Not Allowed`.

Explanation

The following error occurs when running the `prepareserver.sh` script to push the image to the IBM Containers registry.

```
Pushing the MobileFirst Server image to the IBM Containers registry..
    Error response from daemon: <!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
    <title>405 Method Not Allowed</title>
    <h1>Method Not Allowed</h1>
    <p>The method is not allowed for the requested URL.</p>
```

This error typically occurs if the Docker variables have been modified on the host environment. After executing the `initenv.sh` script, the tooling provides an option to override the local docker environment to connect to IBM Containers using native docker commands.

How to resolve

Do not modify the Docker variables (such as DOCKER_HOST and DOCKER_CERT_PATH) to point to the IBM Containers registry environment. For the `prepareserver.sh` script to work correctly, the Docker variables must point to the local Docker environment.

Applying IBM MobileFirst Platform Foundation for iOS interim fixes in an IBM Containers environment

Most interim fixes for IBM MobileFirst Platform Foundation for iOS V7.1 on IBM Containers can be obtained from IBM Fix Central (http://www.ibm.com/support/fixcentral).

Before you begin

Before you apply an interim fix, back up your existing configuration files. The configuration files are located in the *package_root*`/mfpf-analytics/usr` and *package_root*`/mfpf-server/usr` folders.

Procedure

1. After you back up the existing configuration files, download the interim fix archive and extract the contents to your existing installation folder, overwriting the existing files.
2. Restore your backed-up configuration files into the `/mfpf-analytics/usr` and `/mfpf-server/usr` folders, overwriting the newly installed configuration files.

Results

The interim fix has been applied successfully. You can now build and deploy new production-level containers.

Related links:
- IBM MobileFirst Platform Foundation V7.1 products
- IBM Passport Advantage
- IBM Fix Central

Resolving problems

When you are unable to resolve a problem encountered while working with IBM MobileFirst Platform Foundation for iOS on IBM Containers, be sure to gather this key information before contacting IBM Support.

To help expedite the troubleshooting process, gather the following information:
- The version of IBM MobileFirst Platform Foundation that you are using (must be V7.1 or later) and any interim fixes that were applied.

- The container size selected. For example, *Medium 2GB*.
- The Bluemix database plan type. For example, *sqldb_premium*.
- The container ID
- The public IP address (if assigned)
- The information returned from running the following Cloud Foundry CLI plug-in for IBM Containers (**`cf ic`**) commands from the organization and space where your IBM MobileFirst Platform Foundation for iOS container is deployed:

  `cf ic info`

  `cf ic ps -a` (If more than one container instance is listed, make sure to indicate the one with the problem.)
- If Secure Shell (SSH) and volumes were enabled during container creation (while running the `startServer.sh` script), collect all files in the following folders:

  `/opt/ibm/wlp/usr/servers/worklight/logs`

  `/var/log/rsyslog/syslog`
- If only volume was enabled and SSH was not, collect the available log information using the Bluemix dashboard. After you click on the container instance in the Bluemix dashboard, click the **Monitoring and Logs** link in the sidebar. Go to the Logging tab and then click ADVANCED VIEW. The Kibana dashboard opens separately. Using the search toolbar, search for the exception stack trace and then collect the complete details of the exception, *@time-stamp*, *_id*.

Administering MobileFirst applications

Run and maintain MobileFirst applications in production.

IBM MobileFirst Platform Foundation for iOS provides several ways to administer MobileFirst applications in development or in production. MobileFirst Operations Console is the main tool with which you can monitor all deployed MobileFirst applications from a centralized web-based console.

The main operations that you can perform through MobileFirst Operations Console are:

- Deploy mobile applications and adapters to MobileFirst 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. MobileFirst Operations Console, and all administration tools, have four different roles defined for administration of MobileFirst applications. The following MobileFirst administration roles are defined:

Monitor
In this role, a user can monitor deployed MobileFirst 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 IBM MobileFirst Platform Foundation for iOS V7.1.0, the predefined MobileFirst Operations 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 MobileFirst Server during installation of MobileFirst Operations Console. See "Installing the MobileFirst Server administration" on page 6-47.

MobileFirst Operations Console can be used to administer several runtime environments, which are issued from several independent MobileFirst projects and are deployed to the same application server or cluster. For more information about deploying a MobileFirst project, see "Deploying an application from development to a test or production environment" on page 11-1.

MobileFirst Operations Console is not the only way to administer MobileFirst applications. IBM MobileFirst Platform Foundation for iOS 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 API Administration Services" on page 10-4.

With this set of REST services, you can perform the same operations that you can do in MobileFirst Operations Console. You can manage applications, adapters, and, for example, upload a new version of an application or disable an old version.

MobileFirst applications can also be administered by using Ant tasks or with the `wladmin` command line tool. See "Administering MobileFirst applications through Ant" on page 12-12 or "Administering MobileFirst applications through the command line" on page 12-37.

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 MobileFirst applications with MobileFirst Operations Console

You can administer MobileFirst applications through the MobileFirst Operations Console by deploying new versions of mobile and desktop apps, by locking apps or denying access, or by displaying notification messages.

In V6.1.0 and earlier versions of the product, MobileFirst Operations Console was deployed in the project WAR file. If two project WAR files were deployed, each one had its own administration console. Starting with V6.2.0, MobileFirst Operations Console is deployed separately and can administer all runtime environments in the same server.

You can start the console by entering one of the following URLs:

- Secure mode for production or test: `https://`*`hostname`*`:`*`secure_port`*`/worklightconsole`
- Development: `http://`*`server_name`*`:`*`port`*`/worklightconsole`

In either case, the list of all runtime environments is always present. Select the runtime environment that you want to administer to access the functions of the MobileFirst Operations Console.

Use the MobileFirst Operations Console to manage your applications.

- To see all the applications that are installed and all the device platforms that are supported.
- To disable specific application versions on specific platforms and force users to upgrade before they continue to use the application. When you implement direct

updates to mobile devices and desktop apps, software updates are pushed directly to application web resources or users' desktops

- To send out notifications to application users and to manage push notifications from defined event sources to applications.
- To install and manage adapters that are used by applications, and to inspect aggregated usage statistics from MobileFirst Server.
- To lock apps to prevent them from being mistakenly updated and to prevent the redeployment of web resources for a particular application.
- To display a notification message on app start to inform users without causing the application to exit.

Note: Deleting an application from MobileFirst Operations Console will remove all push subscriptions on that application as well.
For an easy way to upgrade an application, see "Upgrading a mobile application in MobileFirst Server and the Application Center" on page 12-86.

Locking an application

You can prevent developers or administrators from mistakenly updating an application, by locking it in MobileFirst Operations Console.

Procedure

Locking an application version for a specific mobile environment

1. From the first page of MobileFirst Operations Console, select the application name.
2. Select **Lock this version** for the application version in the relevant environment.

Remotely disabling application connectivity

You can use the Remote Disable procedure to deny a user's access to a certain application version due to phase-out policy or due to security issues encountered in the application.

Before you begin

If you need to use the Remote Disable feature with servers and clusters that experience heavy loads, consider enabling the Remote Disable cache. Enabling the cache can improve performance by reducing how frequently the database is checked to see if an app has been remotely disabled. By default, the cache is disabled. To enable and configure the cache, add the following lines to the MobileFirst project `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 MobileFirst Server and you set `wl.remoteDisable.cache.refreshIntervalInSeconds=1`, the database is accessed 4 times per second to check the remote disable state.

Note: If code that uses the Remote Disable feature accesses a resource that is protected by OAuth authentication, and the client has a valid token, the MobileFirst Server is not called. As a result, the server does not check whether a specific application is disabled. The MobileFirst Server is called when the token expires or when the Remote Disable realm inside the token expires. When the realm expires, the Remote Disable authenticator is invoked and the server checks whether the specific application version is disabled. For more information about token expiration, see “OAuth-based tokens” on page 8-195.

About this task

Using the MobileFirst Operations Console, you can disable access to a specific version of a specific application for a specific mobile environment and provide a custom message to the user.

Procedure

1. To use this Remote Disable feature, from the first page of MobileFirst Operations Console, click the application name and change the access of the application version that must be disabled from **Active** to **Access Disabled**.
2. Add a custom message as shown in the following text:

   ```
   This version is no longer supported. Please upgrade to the next version.
   ```

   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 12-83.
3. Click **Save**.

   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 MobileFirst 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.

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 MobileFirst 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 MobileFirst Platform Foundation for iOS, when you disabled an application using the MobileFirst Operations 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 MobileFirst Operations 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 it displays the **Close** button.

Related tasks:

"Defining administrator messages from MobileFirst Operations Console in multiple languages"
You can set the deny and notification messages from IBM MobileFirst Platform Operations Console in multiple languages.

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. From the first page of MobileFirst Operations Console, click the relevant application name and change the access of the application version from **Active** to **Active, Notifying**.
2. Add a custom message, such as the following text:
   ```
   Server downtime is planned for Saturday 4am to 6am.
   ```
3. Click **Save**.

Results

The message is displayed the next time that the app is started or resumed. The message is displayed only once.

Related tasks:

"Defining administrator messages from MobileFirst Operations Console in multiple languages"
You can set the deny and notification messages from IBM MobileFirst Platform Operations Console in multiple languages.

Defining administrator messages from MobileFirst Operations Console in multiple languages

You can set the deny and notification messages from IBM MobileFirst Platform Operations Console in multiple languages.

About this task

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.

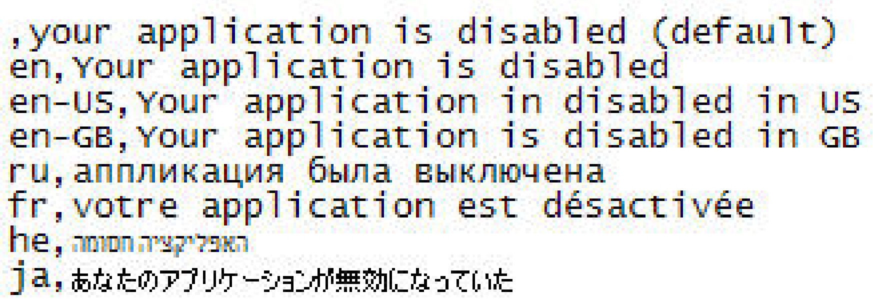
1. From the first page of MobileFirst Operations Console, click the relevant application name and select the access **Active, Notifying** or **Access Disabled**.
2. In the application access section, click “Edit... button”. The label “Supported locales” is followed by the number of available locales. For example, “Supported locales (7)” indicates that seven locales are available for the messages that you want to add.

   In the Messages for multiple languages window, you can upload a CSV file.

   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.

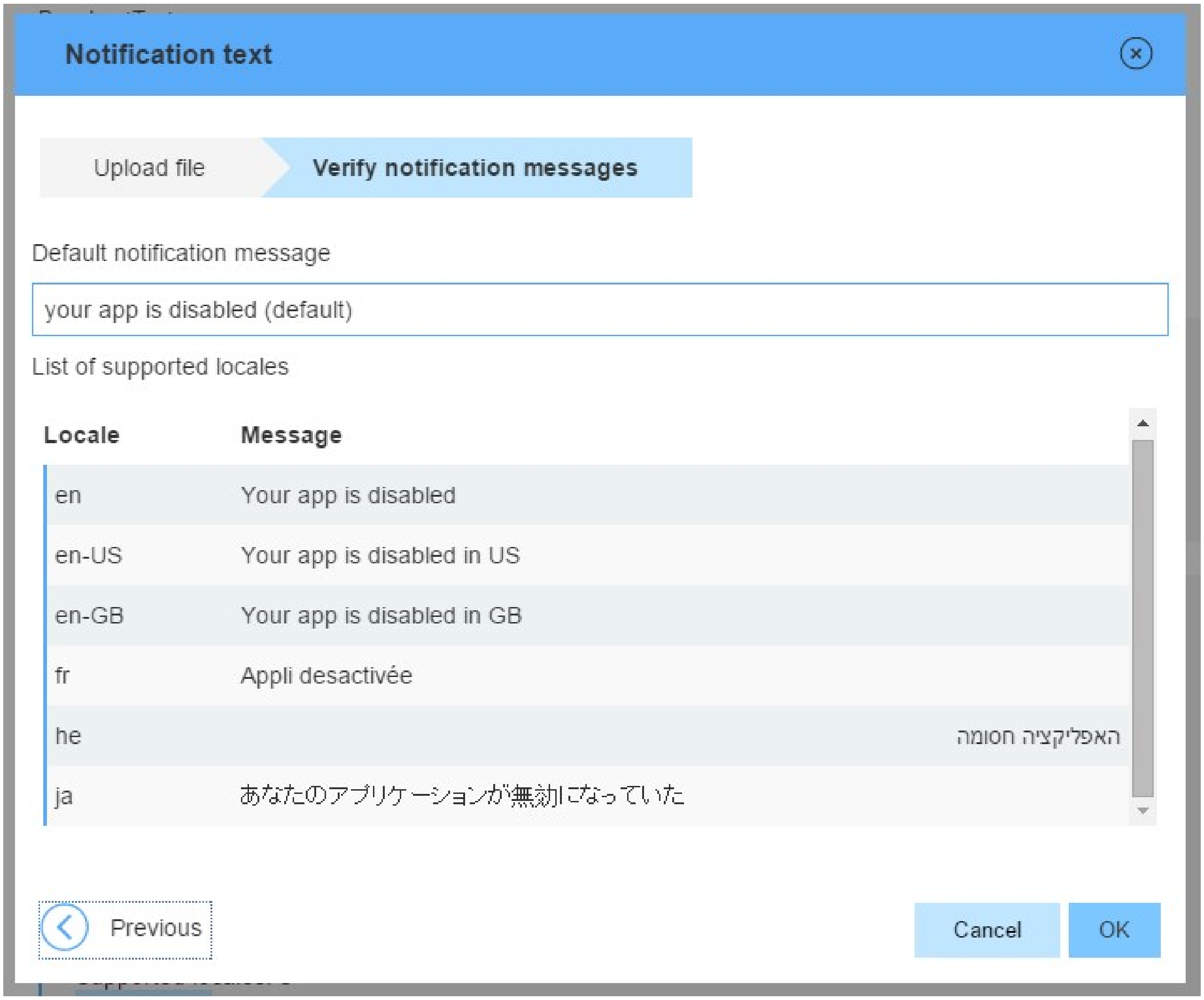
   The following figure shows an example of a CSV file:



```
,your application is disabled (default)
en,Your application is disabled
en-US,Your application in disabled in US
en-GB,Your application is disabled in GB
ru,аппликация была выключена
fr,votre application est désactivée
he,האפליקציה חסומה
ja,あなたのアプリケーションが無効になっていた
```

Figure 12-1. Sample CSV file

3. Click **Upload CSV** to browse and select the CSV file that you want to upload. In “List of supported locales”, you can see the locales and translated messages that you uploaded.



Notification text

Default notification message

your app is disabled (default)

List of supported locales

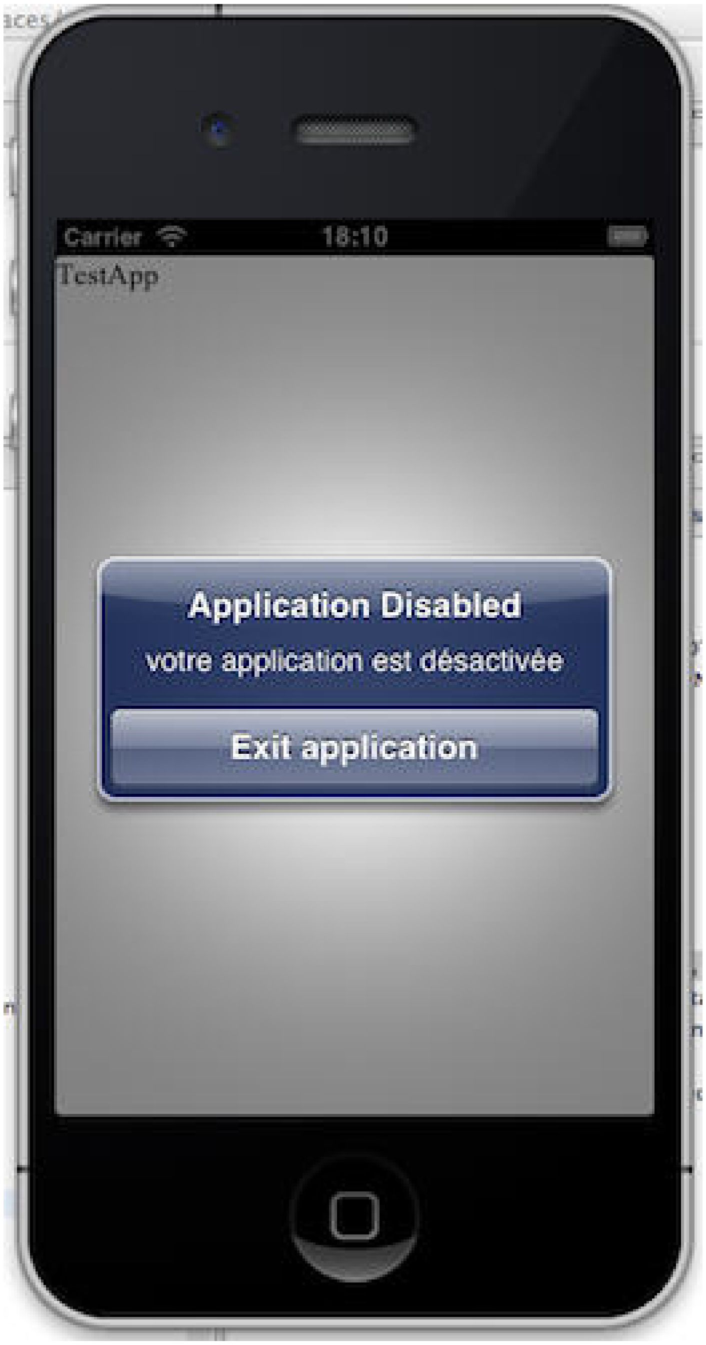
Locale	Message
en	Your app is disabled
en-US	Your app is disabled in US
en-GB	Your app is disabled in GB
fr	Appli desactivée
he	האפליקציה חסומה
ja	あなたのアプリケーションが無効になっていた

Figure 12-2. Verifying locales and notification messages in CSV file

4. Process the uploaded messages:
   - To validate the messages that you uploaded, click **OK**.
   - To discard the changes and return to the console, click **CANCEL**.

   Note: If you modified the default message, then the new default message shows.
5. To clear “List of supported locales”, click **Clear**. This action does not clear the default message.
6. Click **Save** to save the default message and the multilingual messages.

   The following 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.


Carrier 18:10

TestApp

Application Disabled

votre application est désactivée

Figure 12-3. Application Disabled message

Application status and token licensing

You must manually restore the correct application status in MobileFirst Operations Console after `Blocked` status because of insufficient tokens.

If you use token licensing and you no longer have enough license tokens for an application, the application status of all versions of the application changes to `Blocked`. You are no longer able to change the status of any version of the application. The following message is displayed in MobileFirst Operations Console:

```
The application got blocked because its license expired
```

If later enough tokens to run the application become free or your organization purchases more tokens, the following message is displayed in MobileFirst Operations Console:

```
The application got blocked because its license expired but a license is available now
```

The display status is still `Blocked`. You must restore the correct current status manually from memory or your own records by editing the **Status** field. IBM MobileFirst Platform Foundation for iOS does not manage the display of `Blocked` status in MobileFirst Operations Console of an application that was blocked because of insufficient license tokens. You are responsible for restoring such a blocked application to a real status that can be displayed through MobileFirst Operations Console.

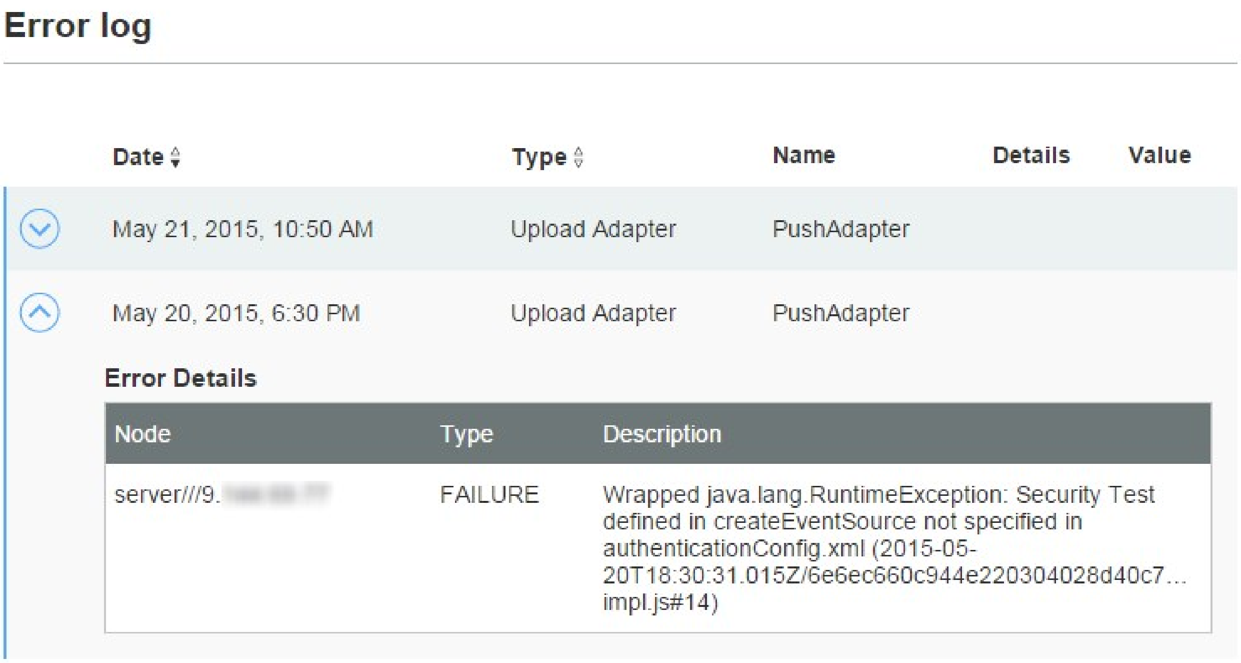
Error log of operations on runtime environments

Use the error log to access failed management operations initiated from MobileFirst Operations Console or the command line on the selected runtime environment, and to see the effect of the failure on the servers.

When a transaction fails, the status bar displays a notification of the error and shows a link to the error log. Use the error log to have more detail about the error, for example, the status of each server with a specific error message, or to have a history of errors. The error log shows the most recent operation first.

You access the error log by clicking **Error log** of a runtime environment in MobileFirst Operations Console.

Expand the row that refers to the failed operation to access more information about the current state of each server. To access the complete log, download the log by clicking **Download log**.



Error log

Date	Type	Name	Details	Value
May 21, 2015, 10:50 AM	Upload Adapter	PushAdapter		
May 20, 2015, 6:30 PM	Upload Adapter	PushAdapter		

Error Details

Node	Type	Description
server///9.[redacted]	FAILURE	Wrapped java.lang.RuntimeException: Security Test defined in createEventSource not specified in authenticationConfig.xml (2015-05-20T18:30:31.015Z/6e6ec660c944e220304028d40c7... impl.js#14)

Figure 12-4. Sample error log

Audit log of administration operations

In the MobileFirst Operations Console, you can refer to an audit log of administration operations.

MobileFirst Operations Console provides access to an audit log for login, logout, and all administration operations, such as deploying apps or adapters or locking apps. The audit log can be disabled by setting the **ibm.worklight.admin.audit** Java Naming and Directory Interface (JNDI) property on the web application of the MobileFirst Administration service (worklightadmin.war) to false.

To access the audit log, click the user name in the header bar and select **About**. Click **Additional support information** and then **Download audit log**.

Each record in the audit log has the following fields, which are separated by a vertical bar (|); see Figure 12-5 on page 12-12.

Table 12-1. Fields in audit log records

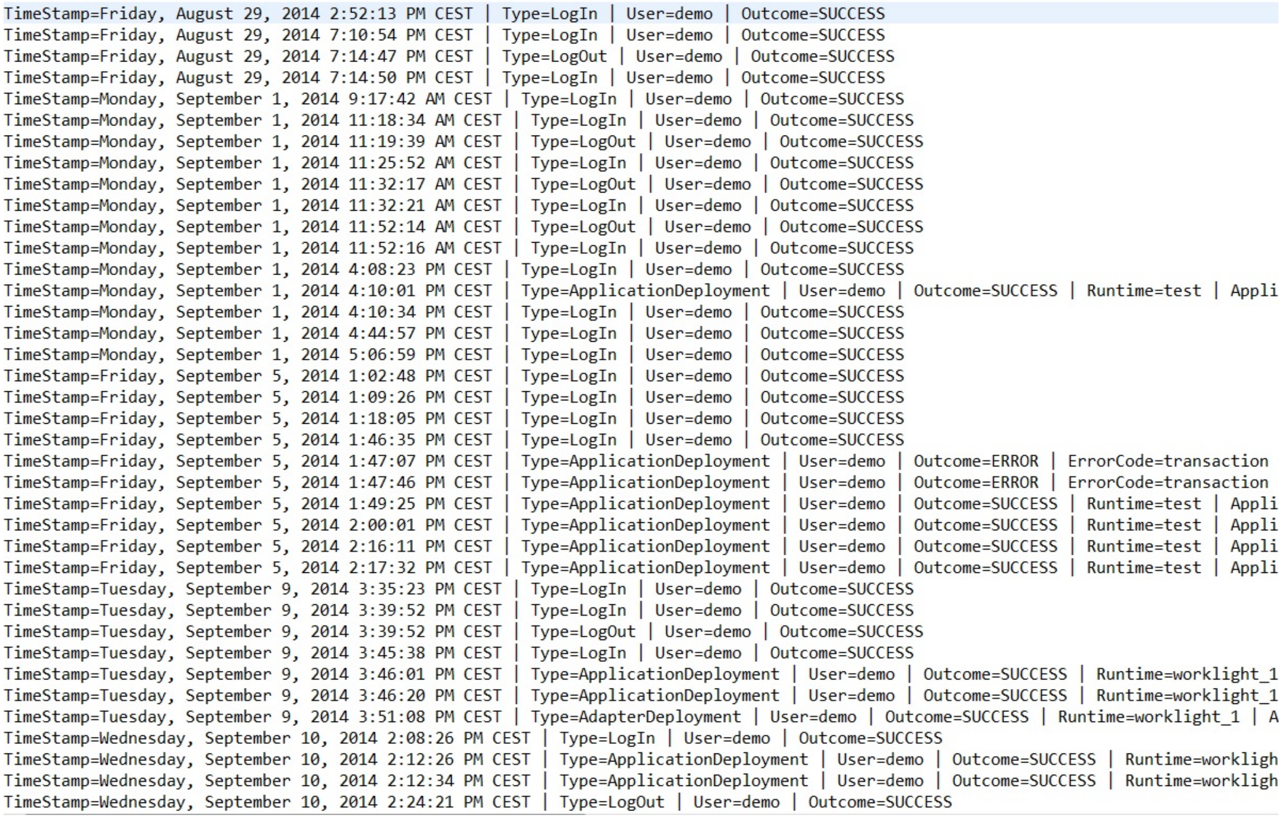
Field name	Description
Timestamp	Date and time when the record was created.
Type	The type of operation. See list of operation types for the possible values.
User	The username of the user who is signed in.
Outcome	The outcome of the operation; possible values are SUCCESS, ERROR, PENDING.
ErrorCode	If the outcome is ERROR, **ErrorCode** indicates what the error is.

Table 12-1. Fields in audit log records (continued)

Field name	Description
Runtime	Name of the MobileFirst project that is associated with the operation.

The following list shows the possible values of **Type** of operation.

- `Login`
- `Logout`
- `AdapterDeployment`
- `AdapterDeletion`
- `ApplicationDeployment`
- `ApplicationDeletion`
- `ApplicationLockChange`
- `ApplicationAuthenticityCheckRuleChange`
- `ApplicationAccessRuleChange`
- `ApplicationVersionDeletion`
- `add config profile`
- `DeviceStatusChange`
- `DeviceApplicationStatusChange`
- `DeviceDeletion`
- `unsubscribeSMS`
- `DeleteDevice`
- `DeleteSubscriptions`
- `SetPushEnabled`
- `SetGCMCredentials`
- `DeleteGCMCredentials`
- `sendMessage`
- `sendMessages`
- `setAPNSCredentials`
- `DeleteAPNSCredentials`
- `setMPNSCredentials`
- `deleteMPNSCredentials`
- `createTag`
- `updateTag`
- `deleteTag`
- `add runtime`
- `delete runtime`



```
TimeStamp=Friday, August 29, 2014 2:52:13 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, August 29, 2014 7:10:54 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, August 29, 2014 7:14:47 PM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
TimeStamp=Friday, August 29, 2014 7:14:50 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 9:17:42 AM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:18:34 AM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:19:39 AM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:25:52 AM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:32:17 AM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:32:21 AM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:52:14 AM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 11:52:16 AM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 4:08:23 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 4:10:01 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=test | Appli
TimeStamp=Monday, September 1, 2014 4:10:34 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 4:44:57 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Monday, September 1, 2014 5:06:59 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, September 5, 2014 1:02:48 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, September 5, 2014 1:09:26 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, September 5, 2014 1:18:05 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, September 5, 2014 1:46:35 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Friday, September 5, 2014 1:47:07 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=ERROR | ErrorCode=transaction
TimeStamp=Friday, September 5, 2014 1:47:46 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=ERROR | ErrorCode=transaction
TimeStamp=Friday, September 5, 2014 1:49:25 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=test | Appli
TimeStamp=Friday, September 5, 2014 2:00:01 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=test | Appli
TimeStamp=Friday, September 5, 2014 2:16:11 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=test | Appli
TimeStamp=Friday, September 5, 2014 2:17:32 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=test | Appli
TimeStamp=Tuesday, September 9, 2014 3:35:23 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Tuesday, September 9, 2014 3:39:52 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Tuesday, September 9, 2014 3:39:52 PM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
TimeStamp=Tuesday, September 9, 2014 3:45:38 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Tuesday, September 9, 2014 3:46:01 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=worklight_1
TimeStamp=Tuesday, September 9, 2014 3:46:20 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=worklight_1
TimeStamp=Tuesday, September 9, 2014 3:51:08 PM CEST | Type=AdapterDeployment | User=demo | Outcome=SUCCESS | Runtime=worklight_1 | A
TimeStamp=Wednesday, September 10, 2014 2:08:26 PM CEST | Type=LogIn | User=demo | Outcome=SUCCESS
TimeStamp=Wednesday, September 10, 2014 2:12:26 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=worklig
TimeStamp=Wednesday, September 10, 2014 2:12:34 PM CEST | Type=ApplicationDeployment | User=demo | Outcome=SUCCESS | Runtime=worklig
TimeStamp=Wednesday, September 10, 2014 2:24:21 PM CEST | Type=LogOut | User=demo | Outcome=SUCCESS
```

Figure 12-5. Sample audit log of MobileFirst administration operations

Administering MobileFirst applications through Ant

You can administer MobileFirst applications through the **wladm** Ant task.

Comparison with other facilities

You can execute administration operations with IBM MobileFirst Platform Foundation for iOS in the following ways:

- The MobileFirst Operations Console, which is interactive.
- The **wladm** Ant task.
- The `wladm` program.
- The MobileFirst administration REST services.

The **wladm** Ant task, `wladm` program, and REST services are useful for automated or unattended execution of operations, such as eliminating operator errors in repetitive operations or operating 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 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" on page 2-9.

For convenience, Apache Ant 1.8.4 is included in MobileFirst Server. In the *product_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.

You can use the **wladm** Ant task on a different computer than the one on which you installed MobileFirst Server.

- Copy the file *product_install_dir*/WorklightServer/worklight-ant-deployer.jar to the computer.
- Make sure that a supported version of Apache Ant and a Java runtime environment are installed on the computer.

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="product_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 by defaults.properties is 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="product_install_dir/WorklightServer/worklight-ant-deployer.jar"/>
  </classpath>
</taskdef>
```

Calling the wladm Ant task

You can use the **wladm** Ant task and its associated commands to administer MobileFirst applications.

Syntax

Call the **wladm** Ant task as follows:

```
<wladm url=... user=... password=...|passwordfile=... [secure=...]>
    some commands
</wladm>
```

Attributes

The **wladm** Ant task has the following attributes:

Table 12-2. List of <wladm> attributes.

Attribute	Description	Required	Default
`url`	The base URL of the MobileFirst web application for administration services	Yes	
`secure`	Whether to avoid operations with security risks	No	`true`
`user`	The user name for accessing the MobileFirst administration services	Yes	
`password`	The password for the user	Either one is required	
`passwordfile`	The file that contains the password for the user		
`timeout`	Timeout for the entire REST service access, in seconds	No	
`connectTimeout`	Timeout for establishing a network connection, in seconds	No	
`socketTimeout`	Timeout for detecting the loss of a network connection, in seconds	No	
`connectionRequestTimeout`	Timeout for obtaining an entry from a connection request pool, in seconds	No	

url

The base URL preferably uses the HTTPS protocol. For example, if you use default ports and context roots, use the following URL.

- For WebSphere Application Server: `https://server:9443/worklightadmin`
- For Tomcat: `https://server:8443/worklightadmin`

secure The default value is `true`. Setting `secure="false"` might have the following effects:

- The user and password might be transmitted in an unsecured 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 host name than the specified server's host name.

password

Specify the password either in the Ant script, through the **`password`** attribute, or in a separate file that you pass 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 knowing this password. To secure the password, before you enter the password into a file, 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.

The **wladm** call contains commands that are encoded in inner elements. These commands 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** call fails.

Elements

You can use the following elements in **wladm** calls:

Table 12-3. 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-beacons	Lists the beacons	0..∞
set-beacon	Specifies information about a beacon	0..∞
show-beacon	Shows information about a beacon	0..∞
remove-beacon	Removes information about a beacon	0..∞
list-beacon-triggers	Lists the beacon triggers	0..∞
set-beacon-trigger	Specifies a beacon trigger	0..∞
show-beacon-trigger	Shows a beacon trigger	0..∞
delete-beacon-trigger	Deletes a beacon trigger	0..∞
list-beacon-trigger-associations	Lists the associations between beacons and beacon triggers	0..∞

Table 12-3. Elements that can be used in <wladm> (continued).

Element	Description	Count
`set-beacon-trigger-association`	Specifies an association between a beacon and a beacon trigger	0..∞
`show-beacon-trigger-association`	Shows the association between a beacon and a beacon trigger	0..∞
`delete-beacon-trigger-association`	Deletes the association between a beacon and a beacon trigger	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 to specific commands, such as `<set-accessrule>`, is in XML too. You can find the XML schemas of these XML formats in the *`product_install_dir`*`/WorklightServer/wladm-schemas/` directory. 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 from the **`wladm`** Ant task is encoded in the encoding format of the current locale. On Windows, this encoding format 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 the output goes to a Windows command prompt window (`cmd.exe`), 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 locale is the default locale on Red Hat Linux and OS X. Many other operating systems have the `en_US.UTF-8` locale.
- Or use the attribute `output="`*`some file name`*`"` to redirect the output of a **`wladm`** command to a file.

Commands for adapters

When you call 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 12-4. **list-adapters** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
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 10-16 REST service.

The deploy-adapter command

The **deploy-adapter** command deploys an adapter in a runtime. It has the following attributes:

Table 12-5. **deploy-adapter** *command attributes*

Attribute	Description	Required	Default
file	Binary adapter file (.adapter).	Yes	Not available
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available

Example

```
<deploy-adapter runtime="worklight" file="MyAdapter.adapter"/>
```

This command is based on the "Adapter (POST)" on page 10-13 REST service.

The show-adapter command

The **show-adapter** command shows details about an adapter. It has the following attributes:

Table 12-6. **show-adapter** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of an adapter.	Yes	Not available
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 10-8 REST service.

The `delete-adapter` command

The **delete-adapter** command removes (undeploys) an adapter from a runtime. It has the following attributes:

Table 12-7. **delete-adapter** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of an adapter.	Yes	Not available

Example

```
<delete-adapter runtime="worklight" name="MyAdapter"/>
```

This command is based on the “Adapter (DELETE)” on page 10-5 REST service.

The `adapter` command group

The **adapter** command group has the following attributes:

Table 12-8. **adapter** *command group attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of an adapter.	Yes	Not available

The command supports the following elements:

Table 12-9. **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 12-10. **get-binary** *command attributes*

Attribute	Description	Required	Default
tofile	Name of the output file.	Yes	Not available

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 10-4 REST service.

Commands for apps

When you call the **wladm** Ant task, you can include various commands for apps.

The enable-extended-authenticity command

The **enable-extended-authenticity** command creates a .wlapp file that is based on an original .wlapp file but has extended authenticity checking enabled. It has the following attributes:

Table 12-11. The **enable-extended-authenticity** *command's attributes*

Attribute	Description	Required	Default
srcwlappfile	Original binary app file (.wlapp, not .apk or .ipa)	Yes	Not available
devicefile	Binary mobile app file (.apk, .appx, .ipa, or .xap)	Yes	Not available
destwlappfile	Output binary app file (.wlapp, not .apk or .ipa)	Yes	Not available

Example

```
<enable-extended-authenticity srcwlappfile="myapp-iphone-1.0.wlapp"
    devicefile="MyApp.ipa"
    destwlappfile="myapp-iphone-1.0.extauth.wlapp"/>
```

Note: This command operates locally, without connecting to the server.

For more information about enabling extended app authenticity checking, see “Configuring extended app authenticity checking” on page 11-56.

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 12-12. The **list-apps** *command's attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
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 10-51 REST service.

The deploy-app command

The **deploy-app** command deploys an app (possibly with multiple environments) in a run time. It has the following attributes:

Table 12-13. The **deploy-app** *command's attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
file	Binary app file (.wlapp, not .apk, or .ipa).	Yes	Not available

Example

```
<deploy-app runtime="worklight" file="MyApp-all.wlapp"/>
```

This command is based on the "Application (POST)" on page 10-47 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 12-14. The **show-app** *command's attributes*

Attribute	Description	Required	Default
runtime	Name of the run time web application / MobileFirst project.	Yes	Not available
name	Name of an app.	Yes	Not available
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 10-43 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 12-15. The **delete-app** *command's attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of an app.	Yes	Not available

Example

```
<delete-app runtime="worklight" name="MyApp"/>
```

This command is based on the "Application (DELETE)" on page 10-40 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 12-16. The **delete-app-version** *command's attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of an app.	Yes	Not available
environment	Mobile platform.	Yes	Not available
version	Version of the app.	Yes	Not available

Note: Deleting an application from MobileFirst Operations Console will remove all push subscriptions on that application as well.

Example

```
<delete-app-version runtime="worklight" name="MyApp" environment="iphone"
version="1.1"/>
```

This command is based on the "App Version (DELETE)" on page 10-34 REST service.

The **app-version** command group

The **app-version** command group has the following attributes:

Table 12-17. The **app-version** *command's group attributes*

Attribute	Description	Required	Default
runtime	Name of the run time web application / MobileFirst project.	Yes	Not available
name	Name of an app.	Yes	Not available
environment	Mobile platform.	Yes	Not available
version	Version of the app.	Yes	Not available

It supports the following elements:

Table 12-18. The **app-version** *command's 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..∞

Table 12-18. The **app-version** *command's group elements (continued)*

Element	Description	Count
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 file wlapp for a version of an app. It has the following attributes:

Table 12-19. The **get-binary** *command's attributes*

Attribute	Description	Required	Default
tofile	Name of the output file.	Yes	Not available

Example

```
<app-version runtime="worklight" name="MyApp" environment="iphone" version="1.1">
  <get-binary tofile="/tmp/MyApp.wlapp"/>
</app-version>
```

This command is based on the “Application Binary (GET, HEAD)” on page 10-39 REST service.

The get-accessrule command

The **get-accessrule** command returns the access rule for an app version. It has the following attributes:

Table 12-20. The **get-accessrule** *command's attributes*

Attribute	Description	Required	Default
output	Name of a file in which to store the output.	No	Not applicable
outputproperty	Name of an Ant property in which to store the output.	No	Not applicable

Example

```
<app-version runtime="worklight" name="MyApp" environment="iphone" version="1.1">
  <get-accessrule output="/tmp/MyApp-accessrule.xml"/>
</app-version>
```

This command is based on the “Application (GET)” on page 10-43 REST service.

The set-accessrule command

The **set-accessrule** command changes the access rule for an app version. It has the following attributes:

Table 12-21. The **set-accessrule** *command's attributes*

Attribute	Description	Required	Default
file	Name of the input file.	Yes	Not available

Example

```
<app-version runtime="worklight" name="MyApp" environment="iphone" 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 10-29 REST service.

The get-authenticitycheckrule command

The **get-authenticitycheckrule** command returns the authenticity check rule for an app version. This command is no longer supported with servers of IBM MobileFirst Platform Foundation for iOS V7.1.0 or later. This command is only available with V6.2.0 and V6.3.0. It has the following attributes:

Table 12-22. The **get-authenticitycheckrule** *command's 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="iphone" version="1.1">
  <get-authenticitycheckrule output="/tmp/MyApp-authenticitycheckrule.txt"/>
</app-version>
```

This command is based on the "Application (GET)" on page 10-43 REST service.

The set-authenticitycheckrule command

The **set-authenticitycheckrule** command changes the authenticity check rule for an app version. This command is no longer supported with servers of IBM MobileFirst Platform Foundation for iOS V7.1.0 or later. This command is only available with V6.2.0 and V6.3.0. It has the following attributes:

Table 12-23. The **set-authenticitycheckrule** *command's attributes*

Attribute	Description	Required	Default
action	Action to perform for authenticity checking.	Yes	Not available

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="iphone" version="1.1">
  <set-authenticitycheckrule action="enabled"/>
</app-version>
```

The `get-lock` command

The **get-lock** command returns information about whether an app version is locked or unlocked. It has the following attributes:

Table 12-24. The **get-lock** *command's 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="iphone" version="1.1">
  <get-lock output="/tmp/MyApp-lock.txt"/>
</app-version>
```

This command is based on the "Application (GET)" on page 10-43 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 12-25. The **set-lock** *command's attributes*

Attribute	Description	Required	Default
lock	New lock state.	Yes	Not available

The possible lock values are `true` and `false`.

Example

```
<app-version runtime="worklight" name="MyApp" environment="iphone" version="1.1">
  <set-lock lock="true"/>
</app-version>
```

This command is based on the "App Version Lock (PUT)" on page 10-37 REST service.

Commands for beacons

When you call the **wladm** Ant task, you can include various commands for the beacons and beacon triggers. A beacon is a piece of information that is associated with an iBeacon. A beacon trigger is an action that a mobile device runs in relation to an iBeacon, when there is an association between the beacon and the beacon trigger.

The `list-beacons` command

The **list-beacons** command returns a list of the beacons that match a given UUID and optionally, a given major and minor number. It has the following attributes.

*Table 12-26. **list-beacons** command attributes*

Attribute	Description	Required	Default
uuid	UUID (32 hex digits) of the beacons to search for.	Only if **major** or **minor** are specified	wild
major	Major number of the beacons to search for.	No	wild
minor	Minor number of the beacons to search for.	No	wild
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<list-beacons uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6"/>
```

This command is based on the "Beacons (GET)" on page 10-85 REST service.

The set-beacon command

The **set-beacon** command specifies or updates information about a beacon. It has the following attributes:

*Table 12-27. **set-beacon** command attributes*

Attribute	Description	Required	Default
file	Name of the input file.	Yes	Not available

The input file can be in JSON or XML format. If it is in XML format, it must follow the schema that is given in the file *product_install_dir*/WorklightServer/wladm-schemas/input/beacon.xsd.

Example:

```
<set-beacon file="entrance.xml"/>
```

This command is based on the "Beacons (PUT)" on page 10-88 REST service.

The show-beacon command

The **show-beacon** command shows details about a beacon. It has the following attributes:

*Table 12-28. **show-beacon** command attributes*

Attribute	Description	Required	Default
uuid	UUID (32 hex digits) of the beacon.	Yes	Not available
major	Major number of the beacon.	Yes	Not available
minor	Minor number of the beacon.	Yes	Not available

Table 12-28. **show-beacon** *command attributes (continued)*

Attribute	Description	Required	Default
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<show-beacon uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1"
minor="23"/>
```

This command is based on the "Beacons (GET)" on page 10-85 REST service.

The `remove-beacon` command

The **remove-beacon** command removes (clears) the information about a beacon. It has the following attributes:

Table 12-29. **remove-beacon** *command attributes*

Attribute	Description	Required	Default
uuid	UUID (32 hex digits) of the beacon.	Yes	Not available
major	Major number of the beacon.	Yes	Not available
minor	Minor number of the beacon.	Yes	Not available

Example:

```
<remove-beacon uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1"
minor="23"/>
```

This command is based on the "Beacons (DELETE)" on page 10-82 REST service.

The `list-beacon-triggers` command

The **list-beacon-triggers** command returns the list of beacon triggers, belonging to a given runtime. It has the following attributes:

Table 12-30. **list-beacon-triggers** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<list-beacon-triggers runtime="worklight"/>
```

This command is based on the "Beacon Triggers (GET)" on page 10-71 REST service.

The set-beacon-trigger command

The **set-beacon-trigger** command specifies or updates information about a beacon trigger, belonging to a given runtime. It has the following attributes:

Table 12-31. set-beacon-trigger command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
file	Name of the input file.	Yes	Not available

The input file can be in JSON or XML format. If it is in XML format, it must follow the schema that is given in the file *product_install_dir*/WorklightServer/wladm-schemas/input/beacon-trigger.xsd.

Example:

```
<set-beacon-trigger runtime="worklight" file="entrance-alert.xml"/>
```

This command is based on the "Beacon Triggers (PUT)" on page 10-78 REST service.

The show-beacon-trigger command

The **show-beacon-trigger** command shows details about a beacon trigger, belonging to a given runtime. It has the following attributes:

Table 12-32. show-beacon-trigger command attributes

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of the beacon trigger.	Yes	Not available
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<show-beacon-trigger runtime="worklight" name="entrance-alert"/>
```

This command is based on the "Beacon Trigger (GET)" on page 10-69 REST service.

The `delete-beacon-trigger` command

The **`delete-beacon-trigger`** command deletes a beacon trigger from a given runtime. It has the following attributes:

Table 12-33. **`delete-beacon-trigger`** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
name	Name of the beacon trigger.	Yes	Not available

Example:

```
<delete-beacon-trigger runtime="worklight" name="entrance-alert"/>
```

This command is based on the "Beacon Trigger (DELETE)" on page 10-66 REST service.

The `list-beacon-trigger-associations` command

The **`list-beacon-trigger-associations`** command returns the list of associations between beacons and beacon triggers that match given criteria, belonging to an app in a given runtime. It has the following attributes:

Table 12-34. **`list-beacon-trigger-associations`** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
app	Name of an app.	Yes	Not available
uuid	UUID (32 hex digits) of the beacon.	Only if **major** or **minor** are specified	
major	Major number of the beacon.	No	
minor	Minor number of the beacon.	No	
triggerName	Name of the beacon trigger.	No	
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Examples:

```
<list-beacon-trigger-associations runtime="worklight" app="productguide"/>
```

```
<list-beacon-trigger-associations runtime="worklight" app="productguide"
uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1" minor="23"/>
```

```
<list-beacon-trigger-associations runtime="worklight" app="productguide"
triggerName="entrance-alert"/>
```

This command is based on the "Associate beacons and triggers (GET)" on page 10-59 REST service.

The `set-beacon-trigger-association` command

The **`set-beacon-trigger-association`** command specifies an association between a beacon and a beacon trigger, belonging to an app in a given runtime. It has the following attributes:

Table 12-35. **`set-beacon-trigger-association`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime web application / MobileFirst project.	Yes	Not available
`app`	Name of an app.	Yes	Not available
`uuid`	UUID (32 hex digits) of the beacon.	Yes	Not available
`major`	Major number of the beacon.	Yes	Not available
`minor`	Minor number of the beacon.	Yes	Not available
`triggerName`	Name of the beacon trigger.	Yes	Not available

Example:

```
<set-beacon-trigger-association runtime="worklight" app="productguide"
uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1" minor="23"
triggerName="entrance-alert"/>
```

This command is based on the "Associate beacons and triggers (PUT)" on page 10-62 REST service.

The `show-beacon-trigger-association` command

The **`show-beacon-trigger-association`** command shows an association between a beacon and a beacon trigger, belonging to an app in a given runtime. It has the following attributes:

Table 12-36. **`show-beacon-trigger-association`** *command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime web application / MobileFirst project.	Yes	Not available
`app`	Name of an app.	Yes	Not available
`uuid`	UUID (32 hex digits) of the beacon.	Yes	Not available
`major`	Major number of the beacon.	Yes	Not available

Table 12-36. **show-beacon-trigger-association** *command attributes (continued)*

Attribute	Description	Required	Default
minor	Minor number of the beacon.	Yes	Not available
triggerName	Name of the beacon trigger.	Yes	Not available
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<show-beacon-trigger-association runtime="worklight" app="productguide"
uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1" minor="23"
triggerName="entrance-alert"/>
```

This command is based on the "Associate beacons and triggers (GET)" on page 10-59 REST service.

The delete-beacon-trigger-association command

The **delete-beacon-trigger-association** command deletes an association between a beacon and a beacon trigger from an app in a given runtime. It has the following attributes:

Table 12-37. **delete-beacon-trigger-association** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
app	Name of an app.	Yes	Not available
uuid	UUID (32 hex digits) of the beacon.	Yes	Not available
major	Major number of the beacon.	Yes	Not available
minor	Minor number of the beacon.	Yes	Not available
triggerName	Name of the beacon trigger.	Yes	Not available

Example:

```
<delete-beacon-trigger-association runtime="worklight" app="productguide"
uuid="496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6" major="1" minor="23"
triggerName="entrance-alert"/>
```

This command is based on the "Associate beacons and triggers (DELETE)" on page 10-56 REST service.

Commands for devices

When you call 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 12-38. **list-devices** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
query	A friendly name or user identifier to search for.	No	
output	Name of output file.	No	
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 10-102 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 12-39. **remove-device** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application / MobileFirst project.	Yes	Not available
id	Unique device identifier.	Yes	Not available

Example:

```
<remove-device runtime="worklight" id="496E974CCEDE86791CF9A8EF2E5145B6"/>
```

This command is based on the "Device (DELETE)" on page 10-96 REST service.

The `device` command group

The **`device`** command group has the following attributes:

*Table 12-40. **`device`** command group attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime web application / MobileFirst project.	Yes	Not available
`id`	Unique device identifier.	Yes	Not available

It supports the following elements:

*Table 12-41. **`device`** command group elements*

Element	Description	Count
`set-status`	Changes the status.	0..∞
`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 12-42. **`set-status`** command attributes*

Attribute	Description	Required	Default
`status`	New status.	Yes	Not available

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 10-99 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 12-43. **`set-appstatus`** command attributes*

Attribute	Description	Required	Default
`app`	Name of an app.	Yes	Not available

*Table 12-43. **set-appstatus** command attributes (continued)*

Attribute	Description	Required	Default
status	New status.	Yes	Not available

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 10-92 REST service.

Commands for troubleshooting

The following commands can help investigate problems with the MobileFirst Server web applications.

The show-info command

The **show-info** command shows basic information about the MobileFirst administration services that can be returned without accessing any runtime nor database. This command can be used to test whether the MobileFirst administration services are running at all. It has the following attributes:

*Table 12-44. **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 MobileFirst versions of various components:

- **wladmVersion**: the exact MobileFirst Server version number from which `worklight-ant-deployer.jar` is taken.
- **productVersion**: the exact MobileFirst Server version number from which `worklightadmin.war` is taken.

And for every project WAR file:

- **serverVersion**: the exact MobileFirst Server version number from which `worklight-jee-library.jar` is taken.
- **platformVersion**: the exact version number of the MobileFirst development tools that built the project WAR file.

It has the following attributes:

Table 12-45. **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 (MobileFirst projects). It has the following attributes:

Table 12-46. **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 10-161 REST service.

The show-runtime command

The **show-runtime** command shows information about a given deployed runtime (MobileFirst project). It has the following attributes:

Table 12-47. **show-runtime** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application or MobileFirst project.	Yes	Not available
output	Name of output file.	No	
outputproperty	Name of Ant property for the output.	No	

Example:

```
<show-runtime runtime="worklight"/>
```

This command is based on the “Runtime (GET)” on page 10-149 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 12-48. **`delete-runtime`** command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime web application or MobileFirst project.	Yes	Not available
`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 10-148 REST service.

The `list-farm-members` command

The **`list-farm-members`** command returns a list of the farm member servers on which a given runtime is deployed. It has the following attributes:

*Table 12-49. **`list-farm-members`** command attributes*

Attribute	Description	Required	Default
`runtime`	Name of the runtime web application or MobileFirst project.	Yes	Not available
`output`	Name of output file.	No	
`outputproperty`	Name of Ant property for the output.	No	

Example:

```
<list-farm-members runtime="worklight"/>
```

This command is based on the “Farm topology members (GET)” on page 10-109 REST service.

The `remove-farm-member` command

The **`remove-farm-member`** command removes a server from the list of farm members on which a given runtime is deployed. This command should be used when the server has become unavailable or disconnected. It has the following attributes:

Table 12-50. **remove-farm-member** *command attributes*

Attribute	Description	Required	Default
runtime	Name of the runtime web application or MobileFirst project.	Yes	Not available
serverId	Identifier of the server.	Yes	Not available
force	Force removal of a farm member, even if it is available and connected.	No	false

Example:

```
<remove-farm-member runtime="worklight" serverId="srvlx15"/>
```

This command is based on the “Farm topology members (DELETE)” on page 10-112 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 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}, version=@{version}:"/>
                    <wladm url="${url}" user="${user}" password="${password}" secure="${secure}">
                      <app-version runtime="@{runtime}" name="@{name}" environment="@{environment}" version="@{version}">
                        <get-accessrule/>
                      </app-version>
                    </wladm>
                  </sequential>
                </actions>
              </call>
            </xmltask>
          </sequential>
        </actions>
      </call>
    </xmltask>
  </target>
</project>
```

Administering MobileFirst applications through the command line

You can administer MobileFirst applications through the `wladm` program.

Comparison with other facilities

You can execute administration operations with IBM MobileFirst Platform Foundation for iOS in the following ways:

- The MobileFirst Operations Console, which is interactive.
- The **`wladm`** Ant task.
- The `wladm` program.
- The MobileFirst administration REST services.

The **`wladm`** Ant task, `wladm` program, and REST services are useful for automated or unattended execution of operations, such as eliminating operator errors in repetitive operations or operating 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`** Ant task is that it is easier to integrate when integration with operating system commands is already available. Moreover, it is more suitable to interactive use.

Prerequisites

The **`wladm`** command is provided in the *`product_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.

To use the wladm program, either put the *product_install_dir*/shortcuts/ directory into your PATH environment variable, or reference its absolute file name in each call.

Calling the wladm program

You can use the wladm program to administer MobileFirst applications.

Syntax

Call the wladm program as follows:

```
wladm --url= --user= ... [--passwordfile=...] [--secure=false] some command
```

The wladm program has the following options:

Table 12-51. wladm program options

Option	Type	Description	Required	Default
--url	URL	Base URL of the MobileFirst web application for administration services	Yes	
--secure	Boolean	Whether to avoid operations with security risks	No	true
--user	name	User name for accessing the MobileFirst admin services	Yes	
--passwordfile	file	File containing the password for the user	No	
--timeout	Number	Timeout for the entire REST service access, in seconds	No	
--connect-timeout	Number	Timeout for establishing a network connection, in seconds	No	
--socket-timeout	Number	Timeout for detecting the loss of a network connection, in seconds	No	
--connection-request-timeout	Number	Timeout for obtaining an entry from a connection request pool, in seconds	No	
--verbose		Detailed output	No	

url

The URL preferably uses the HTTPS protocol. For example, if you use default ports and context roots, use this URL:

- For WebSphere Application Server: https://server:9443/wladmin
- For Tomcat: https://server:8443/wladmin

secure

The **--secure** option is set to true by default. Setting it to --secure=false might have the following effects:

- The user and password might be transmitted in an unsecured 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 host name from the server's host name.

password

Specify the password in a separate file that you pass in the **`--passwordfile`** option. In interactive mode (see “Interactive mode” on page 12-40), 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 knowing 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, do not 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` calls contains a command. The following commands are supported.

Table 12-52. 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

Table 12-52. wladm invocation supported commands (continued)

Command	Description
`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 beacons [`*uuid* `[`*major minor*`]]`	Lists the beacons
`set beacon` *file*	Specifies information about a beacon
`show beacon` *uuid major minor*	Shows information about a beacon
`remove beacon` *uuid major minor*	Removes information about a beacon
`list beacon-triggers [`*runtime-name*`]`	Lists the beacon triggers
`set beacon-trigger [`*runtime-name*`]` *file*	Specifies a beacon trigger
`show beacon-trigger [`*runtime-name*`]` *trigger-name*	Shows a beacon trigger
`delete beacon-trigger [`*runtime-name*`]` *trigger-name*	Deletes a beacon trigger
`list beacon-trigger-associations [`*runtime-name*`]` *app-name* `[`*uuid major minor*`] [`*trigger-name*`]`	Lists the associations between beacons and beacon triggers
`set beacon-trigger-association [`*runtime-name*`]` *app-name uuid major minor trigger-name*	Specifies an association between a beacon and a beacon trigger
`show beacon-trigger-association [`*runtime-name*`]` *app-name uuid major minor trigger-name*	Shows the association between a beacon and a beacon trigger
`delete beacon-trigger-association [`*runtime-name*`]` *app-name uuid major minor trigger-name*	Deletes the association between a beacon and a beacon trigger
`list devices [`*runtime-name*`] [--query` *query*`]`	Lists the devices
`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 in the command line. You can then enter commands interactively, one per line.

The **`exit`** command, or end-of-file on standard input (**`Ctrl-D`** on UNIX terminals) terminates `wladm`.

Help commands are also available in this mode. 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 required, you can disable its use through the option `--no-readline`.

The configuration file

You can also store the options in a configuration file, instead of passing them on the command line at every call. When a configuration file is present and the option –`configfile=file` is specified, you can omit the following options:

- `--url=`*`URL`*
- `--secure=`*`boolean`*
- `--user=`*`name`*
- `--passwordfile=`*`file`*
- `--timeout=`*`seconds`*
- `--connect-timeout=`*`seconds`*
- `--socket-timeout=`*`seconds`*
- `--connection-request-timeout=`*`seconds`*
- *runtime-name*

Use these commands to store these values in the configuration file.

Table 12-53. Commands to store values in the configuration file

Command	Comment
`wladm [--configfile=`*`file`*`] config url `*`URL`*	

Table 12-53. Commands to store values in the configuration file (continued)

Command	Comment
`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 timeout` *seconds*	
`wladm [--configfile=`*file*`] config connect-timeout` *seconds*	
`wladm [--configfile=`*file*`] config socket-timeout` *seconds*	
`wladm [--configfile=`*file*`] config connection-request-timeout` *seconds*	
`wladm [--configfile=`*file*`] config runtime` *runtime-name*	

Use this command to list the values that are stored in the configuration file: `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 MobileFirst Platform Server Data\wladm.config`, or `My Documents\IBM Worklight Server Data\wladm.config`

Note: When you do not specify a **`--configfile`** option, the default configuration file is used only in interactive mode and in **`config`** commands. For noninteractive use of the other commands, you must explicitly designate the configuration file if you want to use one.

Important: 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 12-54. 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 complies with the specific schema. You can disable this check by specifying `--xmlvalidation=none`.

Output character set

Normal output that is produced by the `wladm` program is encoded in the encoding format of the current locale. On Windows, this encoding format is "ANSI code page". The effects are as follows:

- Characters outside of this character set are converted to question marks when they are output.
- When the output goes to a Windows command prompt window (`cmd.exe`), non-ASCII characters are incorrectly displayed because such windows assume characters to be encoded in "OEM code page".

To work around this limitation:

- On operating systems other than Windows, use a locale whose encoding is UTF-8. This format is the default locale on Red Hat Linux and OS X. Many other operating systems have a `en_US.UTF-8` locale.
- Or use the **`wladm`** Ant task, with 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 12-55. list adapters command arguments

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst project.

It takes the following options after the object:

Table 12-56. 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 10-16 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 12-57. deploy adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
file	Binary adapter file (.adapter)

Example:

```
deploy adapter worklight MyAdapter.adapter
```

This command is based on the "Adapter (POST)" on page 10-13 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 12-58. show adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
adapter-name	Name of an adapter

It takes the following options after the object:

Table 12-59. 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 10-8 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 12-60. delete adapter command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
adapter-name	Name of an adapter

Example:

```
delete adapter worklight MyAdapter
```

This command is based on the "Adapter (DELETE)" on page 10-5 REST service.

The adapter command prefix

The `adapter` command prefix takes the following arguments before the verb:

Table 12-61. adapter command prefix arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst 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 12-62. 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 10-4 REST service.

Commands for apps

When you invoke the `wladm` program, you can include various commands for apps.

The `enable extended-authenticity` command

The **`enable extended-authenticity`** command creates a `.wlapp` file that is based on an original `.wlapp` file. But, this new file's extended authenticity property is enabled.

Syntax:

```
enable extended-authenticity src-wlapp-file device-file > dest-wlapp-file
```

It takes the following arguments:

Table 12-63. The **`enable extended-authenticity`** *command's arguments*

Argument	Description
`src-wlapp-file`	Original binary app file (`.wlapp`, not `.apk`, or `.ipa`)
`device-file`	Binary mobile app file (`.apk`, `.appx`, `.ipa`, or `.xap`)
`dest-wlapp-file`	Output binary app file (`.wlapp`, not `.apk`, or `.ipa`)

Example:

```
enable extended-authenticity myapp-iphone-1.0.wlapp MyApp.ipa > myapp-iphone-1.0.extauth.wlapp
```

If you are invoking this command from the operating system command line, you need to escape the ">" character so that the command interpreter (shell or cmd.exe) does not intercept it. For example:

```
wladm enable extended-authenticity myapp-iphone-1.0.wlapp MyApp.ipa ">" myapp-iphone-1.0.extauth.wlap
```

This command operates locally, without connecting to the server. For more information about enabling extended app authenticity checking, see “Configuring extended app authenticity checking” on page 11-56.

The `list apps` command

The **`list apps`** command returns a list of the apps that are deployed in a run time.

Syntax:

```
list apps [runtime-name]
```

It takes the following arguments:

Table 12-64. The **`list apps`** *command's arguments*

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst project.

It takes the following options after the object:

Table 12-65. The **`list apps`** *command's 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 10-51 REST service.

The `deploy app` command

The **deploy app** command deploys an app (possibly with multiple environments) in a run time.

Syntax:

```
deploy app [runtime-name] file
```

It takes the following arguments:

Table 12-66. The **deploy app** *command's arguments*

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst 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 10-47 REST service.

The `show app` command

The **show app** command shows details about an app in a run time, in particular its environments and versions.

Syntax:

```
show app [runtime-name] app-name
```

It takes the following arguments:

Table 12-67. The **show app** *command's arguments*

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
app-name	Name of an app

It takes the following options after the object:

Table 12-68. The **show app** *command's 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 10-43 REST service.

The `delete app` command

The **`delete app`** command removes (undeploys) an app (from all environments, and all versions) from a run time. Deleting an application from MobileFirst Operations Console will remove all push subscriptions on that application as well.

Syntax:

```
delete app [runtime-name] app-name
```

It takes the following arguments:

Table 12-69. The **`delete app`** *command's arguments*

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst project
`app-name`	Name of an app

Example:

```
delete app worklight MyApp
```

This command is based on the "Application (DELETE)" on page 10-40 REST service.

The `delete app version` command

The **`delete app version`** command removes (undeploys) an app version from a run time.

Syntax:

```
delete app version [runtime-name] app-name environment version
```

It takes the following arguments:

Table 12-70. The **`delete app version`** *command's arguments*

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst project.
`app-name`	Name of an app
`environment`	Mobile platform
`version`	Version of the app

Example:

```
delete app version worklight MyApp iPhone 1.1
```

This command is based on the "App Version (DELETE)" on page 10-34 REST service.

The `app version` command prefix

The **`app version`** command prefix takes the following arguments before the verb:

Table 12-71. The **`app version`** *command's prefix arguments*

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst 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 12-72. The **`app version get binary`** *command's arguments*

Argument	Description	Required	Default
`> tofile`	Name of the output file.	No	Standard output

Example:

```
app version worklight MyApp iPhone 1.1 get binary > /tmp/MyApp.wlapp
```

This command is based on the "Application Binary (GET, HEAD)" on page 10-39 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 iPhone 1.1 get accessrule
```

This command is based on the "Application (GET)" on page 10-43 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 12-73. The **app version set accessrule** *command's arguments*

Argument	Description
file	Name of the input file.

Example:

```
app version worklight MyApp iPhone 1.1 set accessrule /tmp/new-accessrule.xml
```

This command is based on the “App Version Access Rule (PUT)” on page 10-29 REST service.

The app version get authenticitycheckrule command

The **app version get authenticitycheckrule** command returns the authenticity check rule for an app version. This command is no longer supported with servers of IBM MobileFirst Platform Foundation for iOS V7.1.0 or later. This command is only available with V6.2.0 and V6.3.0.

Syntax:

```
app version [runtime-name] app-name environment version get authenticitycheckrule
```

Example:

```
app version worklight MyApp iPhone 1.1 get authenticitycheckrule
```

This command is based on the “Application (GET)” on page 10-43 REST service.

The app version set authenticitycheckrule command

The **app version set authenticitycheckrule** command changes the authenticity check rule for an app version. This command is no longer supported with servers of IBM MobileFirst Platform Foundation for iOS V7.1.0 or later. This command is only available with V6.2.0 and V6.3.0.

Syntax:

```
app version [runtime-name] app-name environment version set authenticitycheckrule action
```

It takes the following arguments after the verb:

Table 12-74. The **app version set authenticitycheckrule** *command's arguments*

Argument	Description
action	Action to perform for authenticity checking

The following actions are possible:

- 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 iPhone 1.1 set authenticitycheckrule enabled
```

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 iPhone 1.1 get lock
```

This command is based on the "Application (GET)" on page 10-43 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 12-75. The **`app version set lock`** *command's arguments*

Argument	Description
`lock`	New lock state.

The possible lock values are `true` and `false`.

Example:

```
app version worklight MyApp iPhone 1.1 set lock true
```

This command is based on the "App Version Lock (PUT)" on page 10-37 REST service.

Commands for beacons

When you call the `wladm` program, you can include various commands for the beacons and beacon triggers. A beacon is a piece of information that is associated with an iBeacon. A beacon trigger is an action that a mobile device executes in relation to an iBeacon, when there is an association between the beacon and the beacon trigger.

The list beacons command

The `list beacons` command returns the list of beacons that match a given UUID and optionally, a given major and minor number.

Syntax:

```
list beacons [uuid [major minor]]
```

It takes the following arguments:

Table 12-76. list beacons command arguments

Argument	Description
uuid	UUID (32 hex digits) of the beacons to search for.
major	Major number of the beacons to search for. Use '_' as a wildcard.
minor	Minor number of the beacons to search for. Use '_' as a wildcard.

It takes the following options after the object:

Table 12-77. list beacons options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
list beacons 496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6
```

This command is based on the “Beacons (GET)” on page 10-85 REST service.

The set beacon command

The `set beacon` command specifies or updates information about a beacon.

Syntax:

```
set beacon file
```

It takes the following arguments:

Table 12-78. set beacon command arguments

Argument	Description
file	Name of the input file.

The input file can be in JSON or XML format. If it is in XML format, it must follow the schema that is given in the file *product_install_dir*/WorklightServer/wladm-schemas/input/beacon.xsd.

Example:

```
set beacon entrance.xml
```

This command is based on the “Beacons (PUT)” on page 10-88 REST service.

The show beacon command

The `show beacon` command shows details about a beacon.

Syntax:

```
show beacon uuid major minor
```

It takes the following arguments:

Table 12-79. show beacon command arguments

Argument	Description
`uuid`	UUID (32 hex digits) of the beacon.
`major`	Major number of the beacon.
`minor`	Minor number of the beacon.

It takes the following options after the object:

Table 12-80. show beacon options

Option	Description
`--xml`	Produce XML output instead of tabular output.

Example:

```
show beacon 496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6 1 23
```

This command is based on the "Beacons (GET)" on page 10-85 REST service.

The remove beacon command

The `remove beacon` command removes (clears) the information about a beacon.

Syntax:

```
remove beacon uuid major minor
```

It takes the following arguments:

Table 12-81. remove beacon command arguments

Argument	Description
`uuid`	UUID (32 hex digits) of the beacon.
`major`	Major number of the beacon.
`minor`	Minor number of the beacon.

Example:

```
remove beacon 496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6 1 23
```

This command is based on the "Beacons (DELETE)" on page 10-82 REST service.

The list beacon-triggers command

The `list beacon-triggers` command returns the list of beacon triggers, belonging to a given runtime.

Syntax:

```
list beacon-triggers [runtime-name]
```

It takes the following arguments:

Table 12-82. list beacon-triggers command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.

It takes the following options after the object:

Table 12-83. list beacon-triggers options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
list beacon-triggers worklight
```

This command is based on the "Beacon Triggers (GET)" on page 10-71 REST service.

The set beacon-trigger command

The `set beacon-trigger` command specifies or updates information about a beacon trigger, belonging to a given runtime.

Syntax:

`set beacon-trigger [`*`runtime-name`*`]` *`file`*

It takes the following arguments:

Table 12-84. set beacon-trigger command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
file	Name of the input file.

The input file can be in JSON or XML format. If it is in XML format, it must follow the schema that is given in the file *`product_install_dir`*`/WorklightServer/wladm-schemas/input/beacon-trigger.xsd`.

Example:

```
set beacon-trigger worklight entrance-alert.xml
```

This command is based on the "Beacon Triggers (PUT)" on page 10-78 REST service.

The show beacon-trigger command

The `show beacon-trigger` command shows details about a beacon trigger, belonging to a given runtime.

Syntax:

`show beacon-trigger [`*`runtime-name`*`]` *`trigger-name`*

It takes the following arguments:

Table 12-85. show beacon-trigger command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
trigger-name	Name of the beacon trigger.

It takes the following options after the object:

Table 12-86. show beacon-trigger options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
show beacon-trigger worklight entrance-alert
```

This command is based on the "Beacon Trigger (GET)" on page 10-69 REST service.

The delete beacon-trigger command

The `delete beacon-trigger` command deletes a beacon trigger from a given runtime.

Syntax:

```
delete beacon-trigger [runtime-name] trigger-name
```

It takes the following arguments:

Table 12-87. delete beacon-trigger command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
trigger-name	Name of the beacon trigger.

Example:

```
delete beacon-trigger worklight entrance-alert
```

This command is based on the "Beacon Trigger (DELETE)" on page 10-66 REST service.

The list beacon-trigger-associations command

The `list beacon-trigger-associations` command returns the list of associations between beacons and beacon triggers that match given criteria, belonging to an app in a given runtime.

Syntax:

```
list beacon-trigger-associations [runtime-name] app-name [uuid major minor] [trigger-name]
```

It takes the following arguments:

Table 12-88. list beacon-trigger-associations command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
app-name	Name of an app.
uuid	UUID (32 hex digits) of the beacon.
major	Major number of the beacon. Use '_' as a wildcard.
minor	Minor number of the beacon. Use '_' as a wildcard.
trigger-name	Name of the beacon trigger.

It takes the following options after the object:

Table 12-89. list beacon-trigger-associations options

Option	Description
--xml	Produce XML output instead of tabular output.

Examples:

```
list beacon-trigger-associations worklight productguide
```

```
list beacon-trigger-associations worklight productguide 496E-974C-CEDE-
8679-1CF9-A8EF-2E51-45B6 1 23
```

```
list beacon-trigger-associations worklight productguide entrance-alert
```

This command is based on the “Associate beacons and triggers (GET)” on page 10-59 REST service.

The set beacon-trigger-association command

The set beacon-trigger-association command specifies an association between a beacon and a beacon trigger, belonging to an app in a given runtime.

Syntax:

```
set beacon-trigger-association [runtime-name] app-name uuid major minor trigger-name
```

It takes the following arguments:

Table 12-90. set beacon-trigger-association command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
app-name	Name of an app.
uuid	UUID (32 hex digits) of the beacon.

Table 12-90. set beacon-trigger-association command arguments (continued)

Argument	Description
major	Major number of the beacon.
minor	Minor number of the beacon.
trigger-name	Name of the beacon trigger.

Example:

```
set beacon-trigger-association worklight productguide 496E-974C-CEDE-8679-
1CF9-A8EF-2E51-45B6 1 23 entrance-alert
```

This command is based on the "Associate beacons and triggers (PUT)" on page 10-62 REST service.

The show beacon-trigger-association command

The show beacon-trigger-association command shows an association between a beacon and a beacon trigger, belonging to an app in a given runtime.

Syntax:

```
show beacon-trigger-association [runtime-name] app-name uuid major minor trigger-name
```

It takes the following arguments:

Table 12-91. show beacon-trigger-association command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
app-name	Name of an app.
uuid	UUID (32 hex digits) of the beacon.
major	Major number of the beacon.
minor	Minor number of the beacon.
trigger-name	Name of the beacon trigger.

It takes the following options after the object:

Table 12-92. show beacon-trigger-association options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
show beacon-trigger-association worklight productguide 496E-974C-CEDE-8679-
1CF9-A8EF-2E51-45B6 1 23 entrance-alert
```

This command is based on the "Associate beacons and triggers (GET)" on page 10-59 REST service.

The delete beacon-trigger-association command

The `delete beacon-trigger-association` command deletes an association between a beacon and a beacon trigger from an app in a given runtime.

Syntax:

```
delete beacon-trigger-association [runtime-name] app-name uuid major minor trigger-name
```

It takes the following arguments:

Table 12-93. delete beacon-trigger-association command arguments

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst project.
`app-name`	Name of an app.
`uuid`	UUID (32 hex digits) of the beacon.
`major`	Major number of the beacon.
`minor`	Minor number of the beacon.
`trigger-name`	Name of the beacon trigger.

Example:

```
delete beacon-trigger-association worklight productguide
496E-974C-CEDE-8679-1CF9-A8EF-2E51-45B6 1 23 entrance-alert
```

This command is based on the “Associate beacons and triggers (DELETE)” on page 10-56 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 12-94. list devices command arguments

Argument	Description
`runtime-name`	Name of the runtime web application / MobileFirst 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 12-95. 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 10-102 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 12-96. remove device command arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst project.
id	Unique device identifier.

Example:

```
remove device worklight 496E974CCEDE86791CF9A8EF2E5145B6
```

This command is based on the "Device (DELETE)" on page 10-96 REST service.

The device command prefix

The device command prefix takes the following arguments before the verb:

Table 12-97. device command prefix arguments

Argument	Description
runtime-name	Name of the runtime web application / MobileFirst 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 12-98. 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 10-99 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 12-99. 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 10-92 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 MobileFirst administration services that can be returned without accessing any runtime nor database. This command can be used to test whether the MobileFirst administration services are running at all.

Syntax:

```
show info
```

It takes the following options after the object:

Table 12-100. 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 MobileFirst versions of various components:

- wladmVersion: the exact MobileFirst Server version number from which worklight-ant-deployer.jar is taken.
- productVersion: the exact MobileFirst Server version number from which worklightadmin.war is taken

and for every project WAR file:

- serverVersion: the exact MobileFirst Server version number from which worklight-jee-library.jar is taken
- platformVersion: the exact version number of the MobileFirst development tools that built the project WAR file

Syntax:

```
show versions
```

It takes the following options after the object:

Table 12-101. 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 (MobileFirst projects).

Syntax:

```
list runtimes [--in-database]
```

It takes the following options:

Table 12-102. 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 10-161 REST service.

The show runtime command

The `show runtime` command shows information about a given deployed runtime (MobileFirst project).

Syntax:

```
show runtime [runtime-name]
```

It takes the following arguments:

Table 12-103. show runtime arguments

Argument	Description
runtime-name	Name of the runtime web application or MobileFirst project.

It takes the following options after the object:

Table 12-104. 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 10-149 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 12-105. delete runtime arguments

Argument	Description
runtime-name	Name of the runtime web application or MobileFirst 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 10-148 REST service.

The list farm-members command

The `list farm-members` command returns a list of the farm member servers on which a given runtime is deployed.

Syntax:

```
list farm-members [runtime-name]
```

It takes the following arguments:

Table 12-106. list farm-members arguments

Argument	Description
runtime-name	Name of the runtime web application or MobileFirst project.

It takes the following options after the object:

Table 12-107. list farm-members options

Option	Description
--xml	Produce XML output instead of tabular output.

Example:

```
list farm-members worklight
```

This command is based on the “Farm topology members (GET)” on page 10-109 REST service.

The remove farm-member command

The `remove farm-member` command removes a server from the list of farm members on which a given runtime is deployed. This command should be used when the server has become unavailable or disconnected.

Syntax:

```
remove farm-member [runtime-name] server-id
```

It takes the following arguments:

Table 12-108. remove farm-member arguments

Argument	Description
`runtime-name`	Name of the runtime web application or MobileFirst project.
`server-id`	Identifier of the server.

It takes the following options after the object:

Table 12-109. remove farm-member option

Option	Description
`--force`	Force removal of a farm member, even if it is available and connected.

Example:

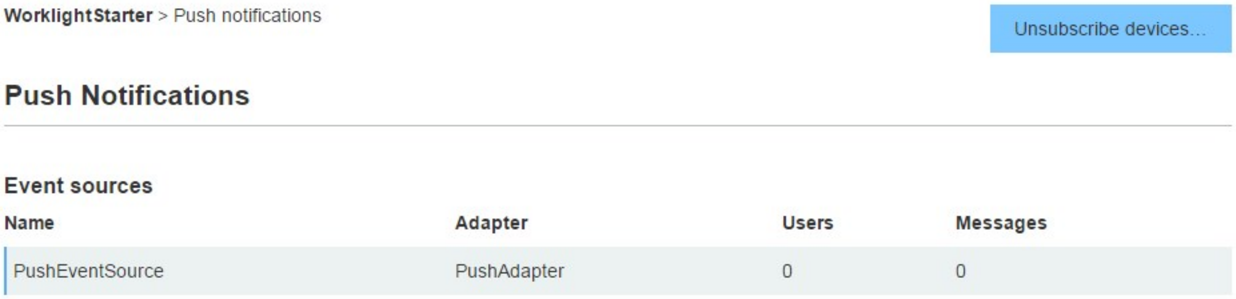
```
remove farm-member worklight srvlx15
```

This command is based on the “Farm topology members (DELETE)” on page 10-112 REST service.

Administering push notifications with the MobileFirst Operations Console

The Push Notifications page in the MobileFirst Operations Console provides you with a quick view of the various entities in the push notification chain.

The **Event Sources** tab displays the list of data sources that are configured in your MobileFirst Server, including the number of users that are subscribed to notifications from each source.



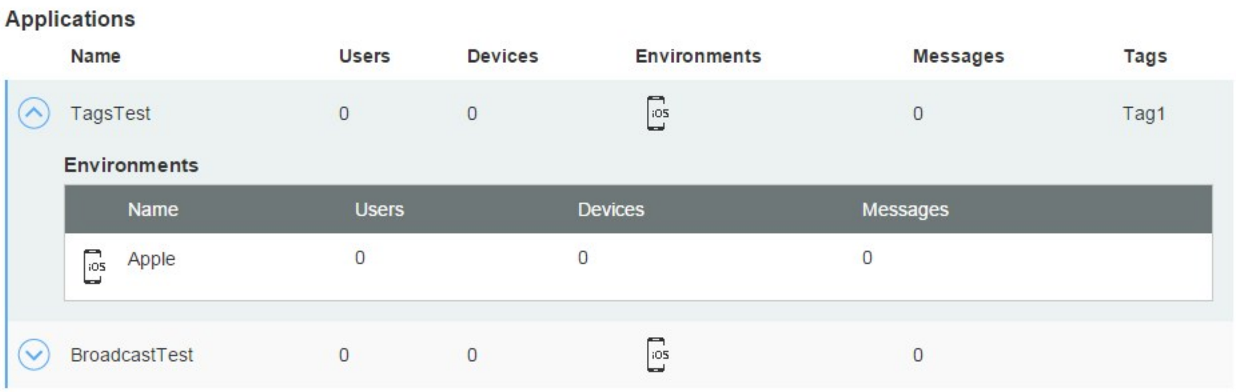
Push Notifications

Event sources

Name	Adapter	Users	Messages
PushEventSource	PushAdapter	0	0

Figure 12-6. Data sources of push notifications in the MobileFirst Operations Console

Clicking **Applications** displays the deployed applications that 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.



Applications

Name	Users	Devices	Environments	Messages	Tags
TagsTest	0	0	iOS	0	Tag1

Environments

Name	Users	Devices	Messages
iOS Apple	0	0	0

Name	Users	Devices	Environments	Messages	Tags
BroadcastTest	0	0	iOS	0	

Figure 12-7. Deployed applications that receive push notifications in the MobileFirst Operations Console

As Administrator, you can forcibly unsubscribe existing SMS subscriptions by clicking **Unsubscribe Devices**. As Administrator, you can enter the mobile phone numbers to be unsubscribed.

Note: You can 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 two subscriptions exist for the same phone number and user name, unsubscribing by using the MobileFirst Operations Console unsubscribes both 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 Application Center

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 Application Center is similar to the concept of the Apple public App Store, except that it targets only private usage within a company.

By using 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.

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. Application Center does not target mobile web applications; such applications are delivered to the mobile device web browser through a URL like a website.

Application Center manages mobile applications; it supports any kind of iOS application, including applications that are built on top of the MobileFirst platform.

You can use the Application Center as part of the development process of an application. A typical scenario of Application Center is a team building a mobile application; the development team creates a new version of an iOS application. The development team wants this new version to be reviewed and tested by the extended team. A developer goes to Application Center console and uploads the new version of the application to 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 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.

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

The iOS operating system imposes specific requirements for deploying, installing, or using applications on mobile devices.

All iOS applications that are managed through 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.

For more information about “Ad Hoc Distribution”, see the Apple page about ad hoc provisioning profiles.

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 12-76.

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 12-105.

The following figure shows an overview of the architecture.

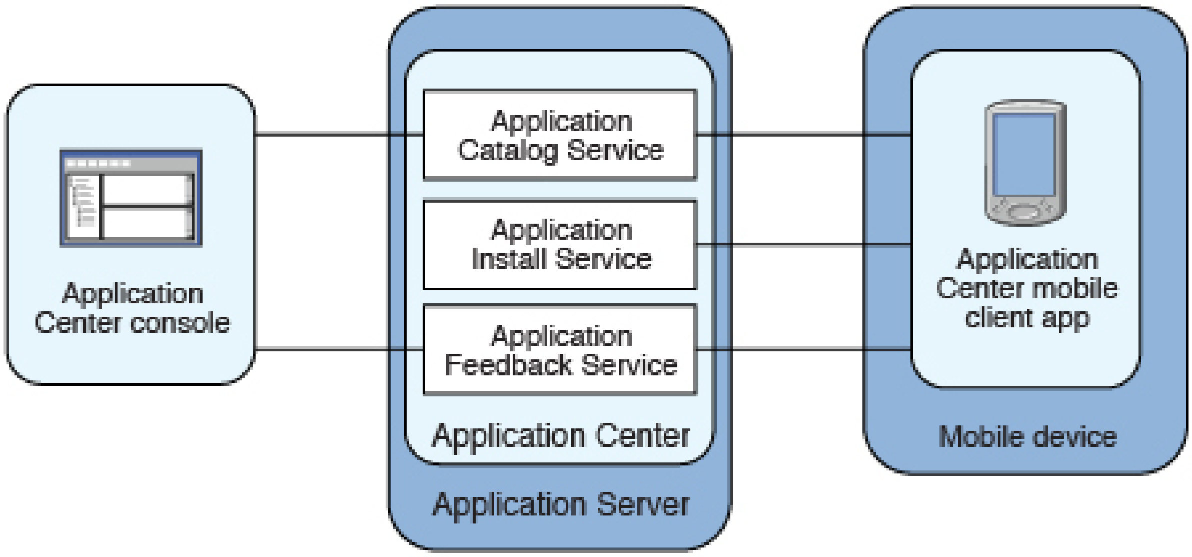

Application Center console
Application Catalog Service
Application Install Service
Application Feedback Service
Application Center
Application Server
Application Center mobile client app
Mobile device

Figure 12-8. 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 who 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.

You will find instructions in this document about how to configure the Application Center server-side component on various Java application servers after IBM MobileFirst Platform Foundation for iOS is installed, as well as how to build MobileFirst 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 MobileFirst Platform Foundation for iOS is installed, start the Application Center console, and log in.

When you install IBM MobileFirst Platform Foundation for iOS, 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 user authentication for Application Center" on page 6-243 or, if you are using LDAP authentication, "Managing users with LDAP" on page 6-248.

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 12-76.

To install and run the mobile client on iOS operating system, see "Installing the client on an iOS mobile device" on page 12-105.

Preparations for using the mobile client

To use the mobile client to install apps on mobile devices, you must first import the **IBMAppCenter** project into the 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 iOS 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 MobileFirst mobile application.

The MobileFirst project **IBMAppCenter** contains the iOS version of the client.

The iOS version for iPad and iPhone is not delivered as a compiled application. The application must be created from the MobileFirst 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 MobileFirst and Apple software. The version of MobileFirst Platform Command Line Interface for iOS must be the same as the version of IBM MobileFirst Platform Server on which this documentation is based. The Apple Xcode version is V6.1.

Importing and building the project

You must import the **IBMAppCenter** project into MobileFirst Studio and then build the project.

About this task

Follow the normal procedure to import a project into MobileFirst Studio.

Application Center requires MobileFirst Studio for importing and building the IBMAppCenter project. MobileFirst Studio is not part of IBM MobileFirst Platform Foundation for iOS, but if you purchased this product, you are entitled to the full cross-platform version of the product as well. For information about how to obtain MobileFirst Studio, see "Application Center requires MobileFirst Studio for importing and building the IBMAppCenter project" on page 3-21.

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 MobileFirst Studio.

What to do next

Build the **IBMAppCenter** project. The MobileFirst project contains a single application named `AppCenter`. Right-click the application and select **Run as** > **Build All Environments**.

iOS MobileFirst 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 12-73 for more information.

Customizing features (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 12-110. 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,
`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.
`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

iOS: Files named `icon`*`size`*`.png` in the `IBMAppCenter/apps/AppCenter/iphone/native/Resources` directory.

Application name

iOS: Edit the **`CFBundleDisplayName`** key in the `IBMAppCenter/apps/AppCenter/iphone/native/IBMAppCenterAppCenterIphone-Info.plist` file.

Splash screen images

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`

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`.

Deploying the mobile client in Application Center

Deploy the different versions of the client application to Application Center.

The iOS version of the mobile client must be deployed to the Application Center. To do so, you must upload the iOS application (.ipa) files to the Application Center.

Follow the steps described in “Adding a mobile application” on page 12-79 to add the mobile client application for iOS. 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.

Related tasks:

“Installing the client on an iOS mobile device” on page 12-105
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.

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 notification automatically, to any iOS device. Notifications are sent for updates to favorite applications and of new applications that are deployed on the Application Center server and that are marked as recommended.

Push notification process

Push notifications are sent to a device if the following conditions are met:

- The device has Application Center installed and started it at least one time.
- The user has not disabled push notification for this device for the Application Center in the **Settings** > **Notifications** interface.
- The user is allowed to install the application. Such permissions are controlled through the Application Center access rights.
- The application is marked as recommended, or is marked as preferred for the user who is using Application Center on this device. Those flags are set automatically when the user installs an application through Application Center. You can see which applications are marked as preferred by looking at the Application Center **Favorites** tab on the device.
- The application is not installed on the device or a more recent version is available than the version that is installed on the device.

  The first time that the Application Center client starts on a device, the user might be asked whether to accept incoming push notifications. This is the case for iOS mobile devices. The push notification feature does not work when the service is disabled on the mobile device.

  Application Center cannot know precisely which app versions are installed because it does not detect what apps the user removed. Hence, if the user installs the latest version of an application but removes it later, no update is notified because the Application Center does not detect the removal. iOS 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.

Related concepts:

"Marking or unmarking a favorite app" on page 12-123
Mark your favorite apps or unmark an app to have it removed from the favorites list.

Related reference:

"Application properties" on page 12-83
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.

Configuring push notifications for application updates

Configure the Application Center services to communicate with 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 6-280 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.Strin
```

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 Apple Push Notification Services

Configure your iOS project for Apple Push Notification Services (APNs).

Before you begin

Ensure that the following servers are accessible from Application Center 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

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.

Note: Devices that are connected to a company wifi behind a firewall are only able to receive push notifications if connection to the following type of address is not blocked by the firewall.

x-courier.sandbox.push.apple.com:5223

Where *x* is an integer number.

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 12-111. JNDI properties

JNDI Property	Type and description	Example for Apache Tomcat server

`ibm.appcenter.apns.p12.certificate.location`	A string value that defines the full path to the `.p12` certificate.	`<Environment name="`*`ibm.appcenter.apns.p12.certificate.loca`*
		`override="false" type="java.lang.String" value=`
		`"/Users/someUser/someDirectory/apache-tomcat/conf/AppCente`

`ibm.appcenter.apns.p12.certificate.password`	A string value that defines the password needed to access the certificate.	`<Environment name="`*`ibm.appcenter.apns.p12.certificate.pass`*
		`type="java.lang.String"`
		`value="`*`this_is_a_secure_password`*`"/>`

`ibm.appcenter.apns.p12.certificate.isDevelopmentCertificate`	A boolean value (identified as true or false) that defines whether or not the provisioning profile used to generate the authentication certificate was a development certificate.	`<Environment name="ibm.appcenter.apns.p12.certificate.isDe`
		`override="false" type="java.lang.String"`
		`value="true"/>`

See "List of JNDI properties for the Application Center" on page 6-280 for a complete list of JNDI properties that you can set.

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 iOS.
- Manage several different versions of mobile applications.
- Review the feedback of testers of mobile applications.
- Define the users who have the rights to list and install an application on the mobile devices.
- Track which applications are installed on which devices.

Note:

Only users with the administrator role can log in to the Application Center console.

Multicultural support: the user interface of the Application Center console has not been translated.

Starting the Application Center console

You can start the Application Center with your web browser and log in if you have the administrator role.

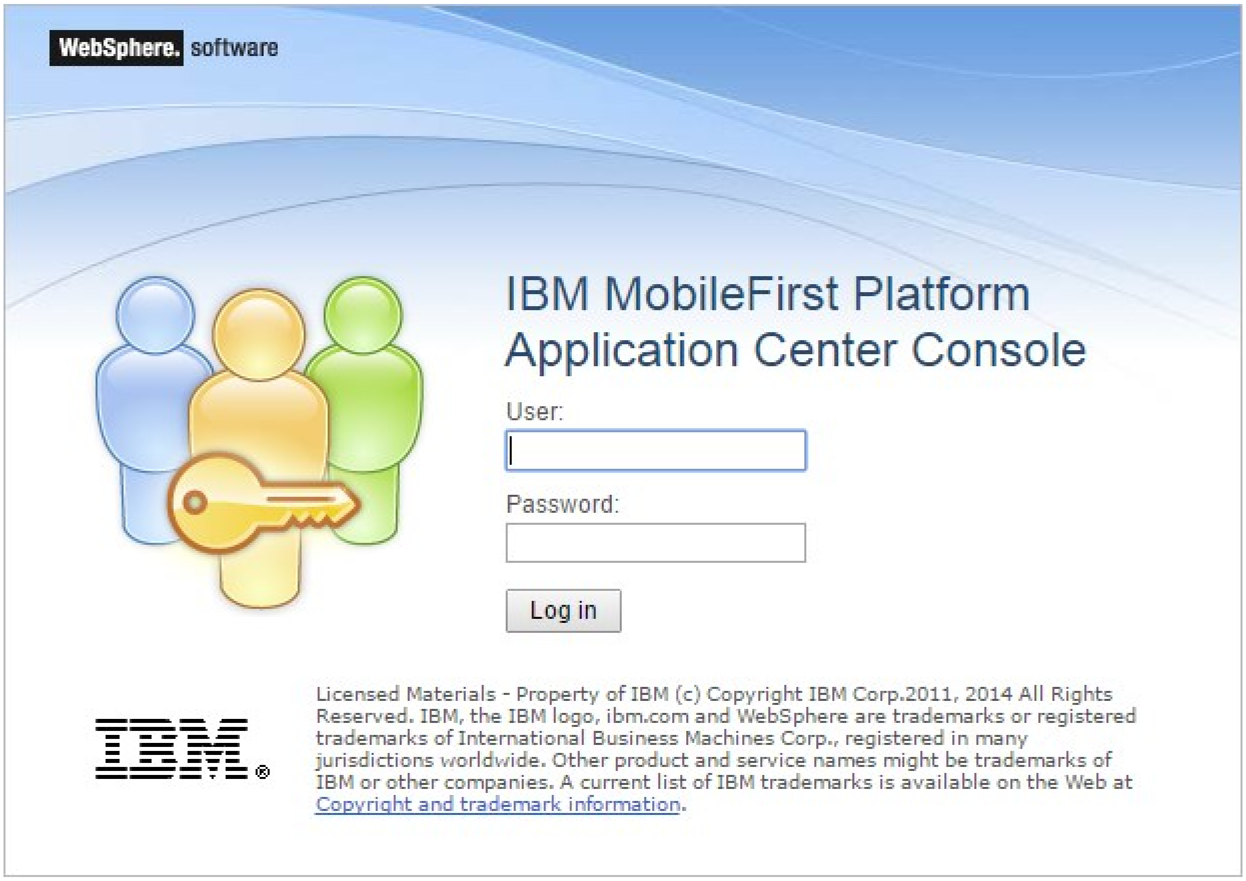
Procedure

1. Start a web browser session on your desktop.
2. Contact your system administrator to obtain the address and port of the server where the Application Center is installed.
3. Enter the following URL: `http://`*`server`*`/appcenterconsole`

   Where *server* is the address and port of the server where the Application Center is installed.

   `http://localhost:9080/appcenterconsole`
4. Log in to the Application Center console

   Contact your system administrator to get your credentials so that you can log in to the Application Center console.



Licensed Materials - Property of IBM (c) Copyright IBM Corp.2011, 2014 All Rights Reserved. IBM, the IBM logo, ibm.com and WebSphere are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information.

Figure 12-9. 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.

Troubleshooting a corrupted login page in Safari browsers

You can recover from a corrupted login page of the Application Center console when you use the Safari browser.

Symptom

When the Application Center console is open in a Safari browser, you might navigate away from the console. When you come back to the console, you might see the login page. Even though you enter the correct login details, you see the following message instead of a successful login:

```
HTTP Status 404 - appcenterconsole/j_security_check.
```

Cause

The behavior is linked to a caching problem in the Safari browser.

Solution

The workaround is to trigger a forced reload when you see the login page without entered or autocompleted credentials. Here is how to trigger a forced reload:

- On a Mac computer, press Shift + the **Refresh** button.
- On an iPad or iPhone device: Double-click the refresh button or clean the cache by closing Safari: you double-click the home button and then swipe Safari away.

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 IBM MobileFirst Platform Foundation for iOS 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.

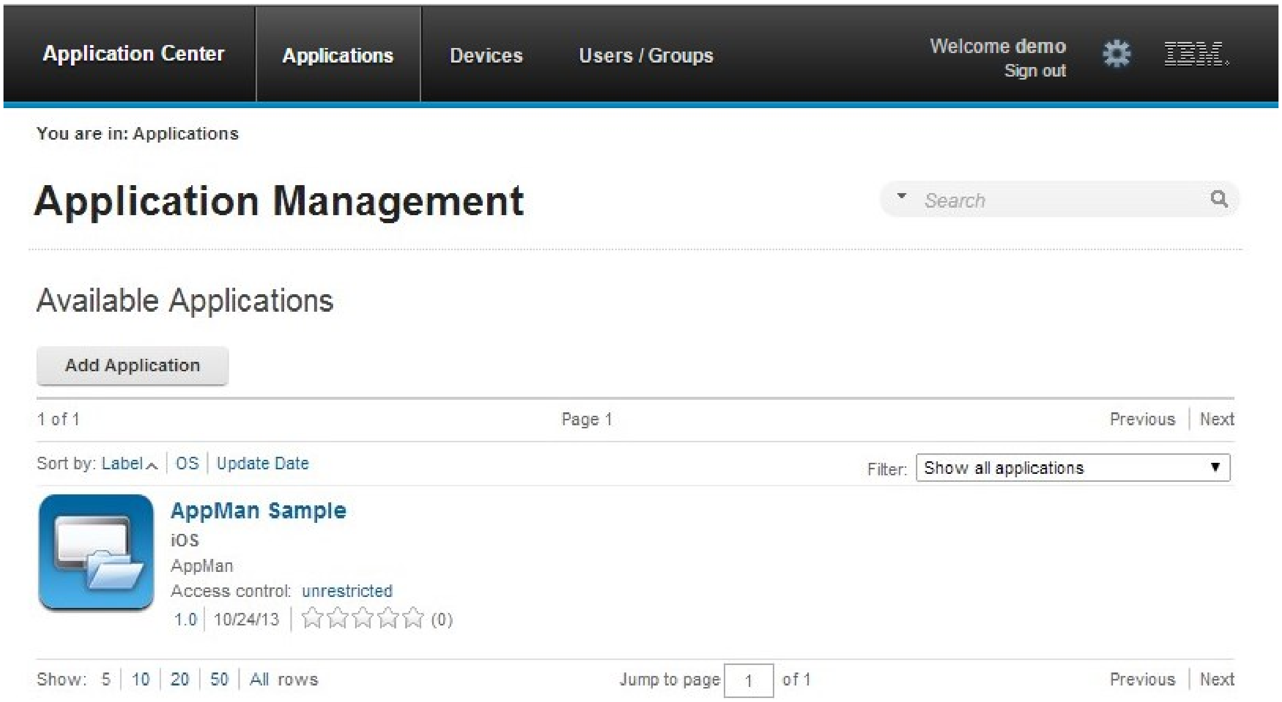


Figure 12-10. Available applications

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 Application Center. Initially the list of applications is empty and you must upload an application file. Application files are described in this procedure.

Procedure

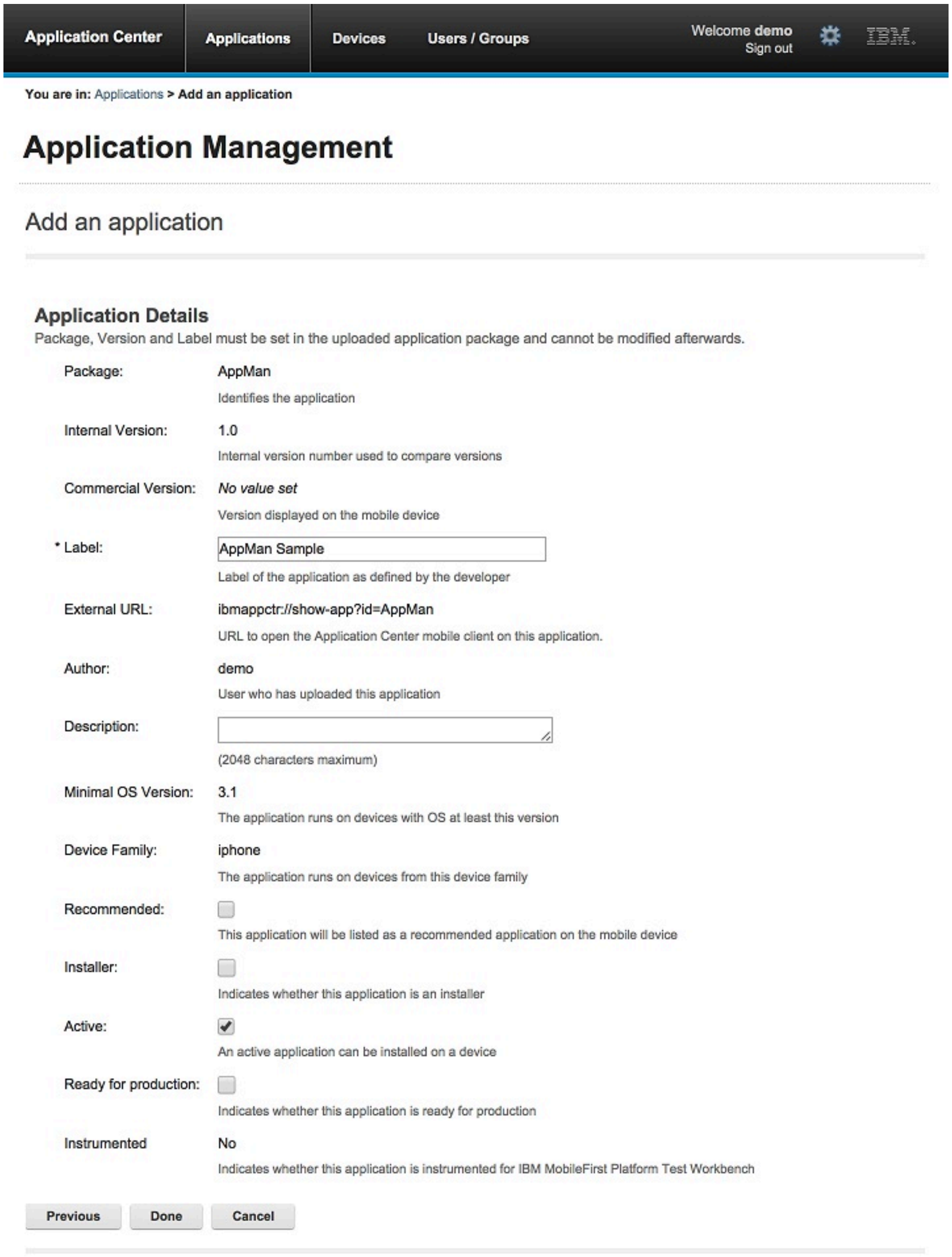
To add an application to make it available to be installed on mobile devices:

1. Click **Add Application**.
2. Click **Upload**.
3. Select the application file to upload to the Application Center repository.

   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 MobileFirst Platform Test Workbench.
4. Click **Next** to access the properties to complete the definition of the application.
5. Complete the properties to define the application. See Application properties for information about how to complete property values.
6. Click **Finish**.



You are in: Applications > Add an application

Application Management

Add an application

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package:	AppMan Identifies the application
Internal Version:	1.0 Internal version number used to compare versions
Commercial Version:	*No value set* Version displayed on the mobile device
* Label:	AppMan Sample Label of the application as defined by the developer
External URL:	ibmappctr://show-app?id=AppMan URL to open the Application Center mobile client on this application.
Author:	demo User who has uploaded this application
Description:	(2048 characters maximum)
Minimal OS Version:	3.1 The application runs on devices with OS at least this version
Device Family:	iphone The application runs on devices from this device family
Recommended:	□ This application will be listed as a recommended application on the mobile device
Installer:	□ Indicates whether this application is an installer
Active:	☑ An active application can be installed on a device
Ready for production:	□ Indicates whether this application is ready for production
Instrumented	No Indicates whether this application is instrumented for IBM MobileFirst Platform Test Workbench

Figure 12-11. 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 12-79.

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 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.
   - **Apple iTunes**:
     a. When the list of items is returned in the search result, select the item that you want.
     b. At the bottom of the selected application, click “Direct Link” to open the application details page.

        Note: Do not copy the “Direct Link” to the Application Center. “Direct Link” is a URL with redirection, you will need to get the URL it redirects to.
     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.
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.

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 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 12-115
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**

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 MobileFirst Platform Test Workbench for IBM MobileFirst Platform Foundation for iOS or a normal iOS application.
- **External URL**Use this URL to have the mobile client of the Application Center started 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 12-105.

Active

Select **Active** to indicate that an application can be installed on a mobile device. If you do not select **Active**, the mobile user does not see the application in the list of available applications that are 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

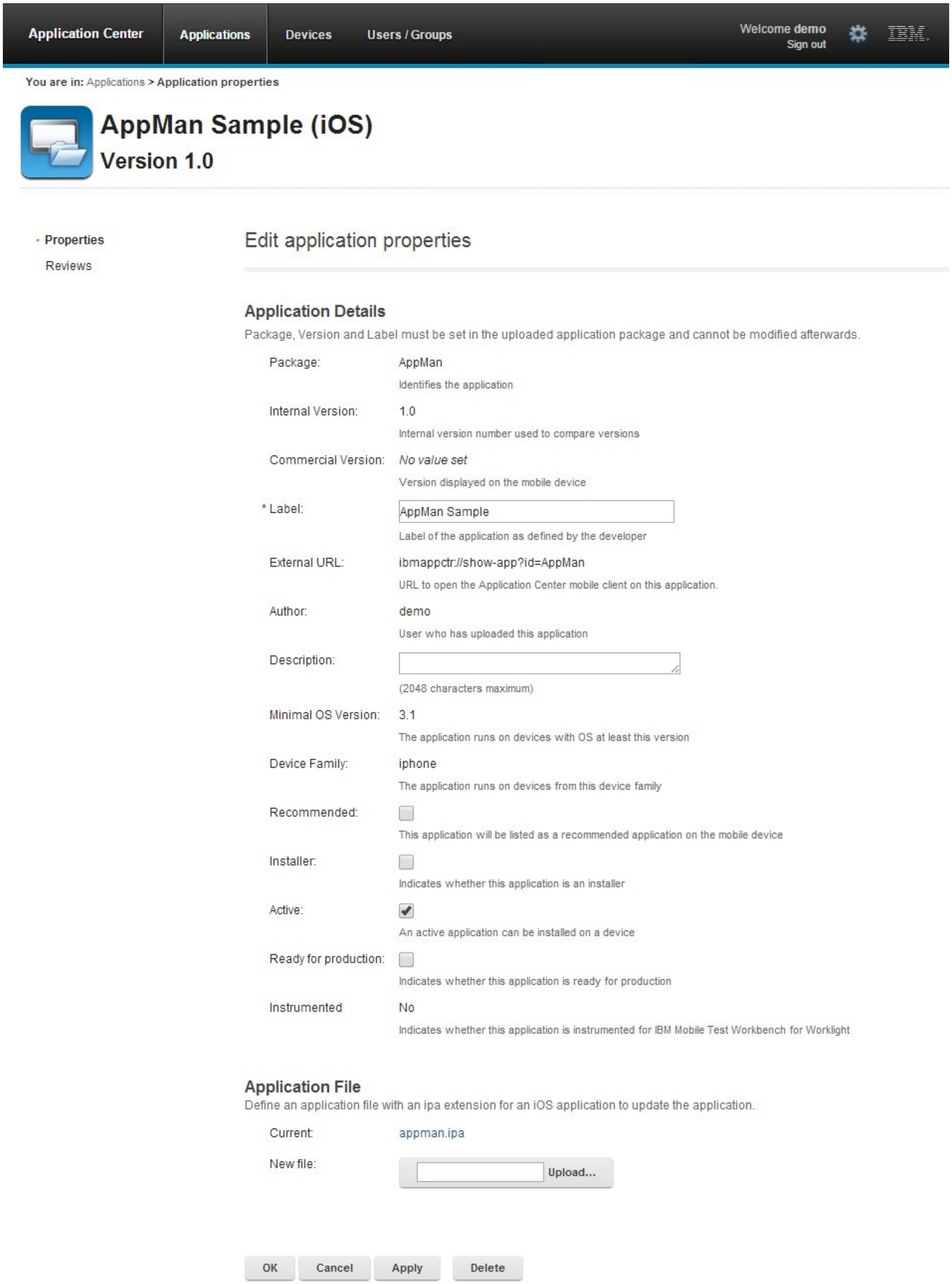
You can edit the properties of an application in the list of uploaded applications.

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 12-83 for details about these properties. The name of the current application file is shown after the properties.

   Important: If you want to update the file, it must belong to the same package and be the same version number. If either of these properties is not the same you must go back to the application list and add the new version first.
4. Click **OK** to save your changes and return to Available Applications or **Apply** to save and keep Application Details open.



You are in: Applications > Application properties

AppMan Sample (iOS)
Version 1.0

Edit application properties

Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package: AppMan
Identifies the application

Internal Version: 1.0
Internal version number used to compare versions

Commercial Version: *No value set*
Version displayed on the mobile device

* Label: AppMan Sample
Label of the application as defined by the developer

External URL: ibmappctr://show-app?id=AppMan
URL to open the Application Center mobile client on this application.

Author: demo
User who has uploaded this application

Description:
(2048 characters maximum)

Minimal OS Version: 3.1
The application runs on devices with OS at least this version

Device Family: iphone
The application runs on devices from this device family

Recommended: □
This application will be listed as a recommended application on the mobile device

Installer: □
Indicates whether this application is an installer

Active: ☑
An active application can be installed on a device

Ready for production: □
Indicates whether this application is ready for production

Instrumented No
Indicates whether this application is instrumented for IBM Mobile Test Workbench for Worklight

Application File

Define an application file with an ipa extension for an iOS application to update the application.

Current: appman.ipa

New file:

Figure 12-12. Application properties for editing

Upgrading a mobile application in MobileFirst Server and the Application Center

You can easily upgrade deployed mobile applications by using a combination of MobileFirst Operations 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 MobileFirst Server when the application is running.

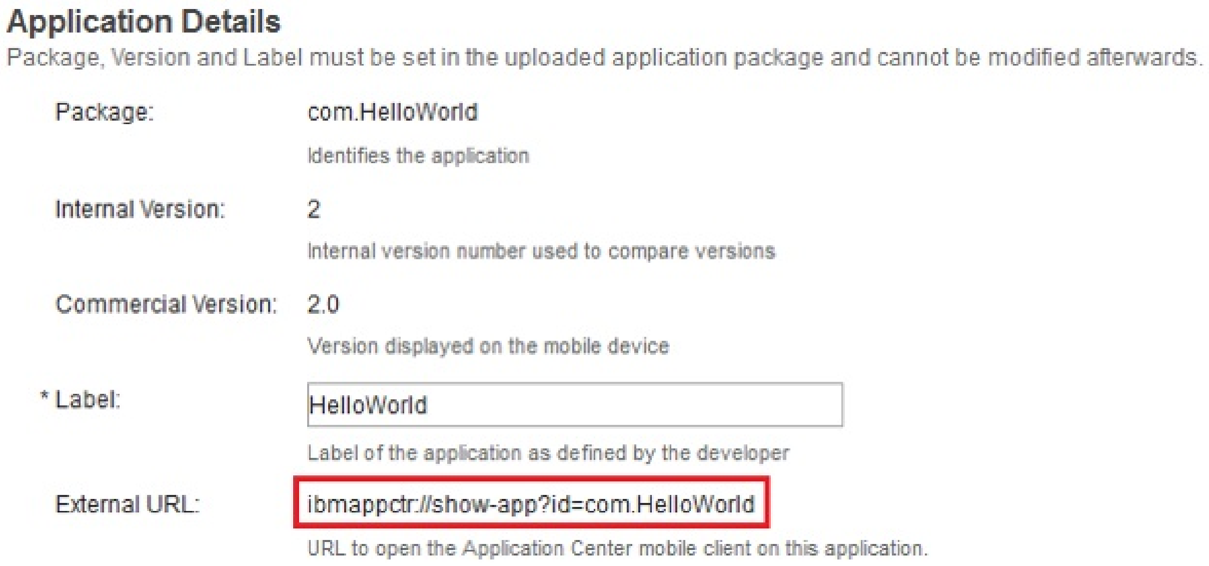
About this task

You can use this procedure to update iOS applications that have been deployed on MobileFirst Server and also in the Application Center. In this task, the application HelloWorld version 1.0 is already deployed on MobileFirst Server and in the Application Center.

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 12-79.
2. From the Application Details page, copy the setting of the external URL.



Application Details

Package, Version and Label must be set in the uploaded application package and cannot be modified afterwards.

Package: com.HelloWorld
Identifies the application

Internal Version: 2
Internal version number used to compare versions

Commercial Version: 2.0
Version displayed on the mobile device

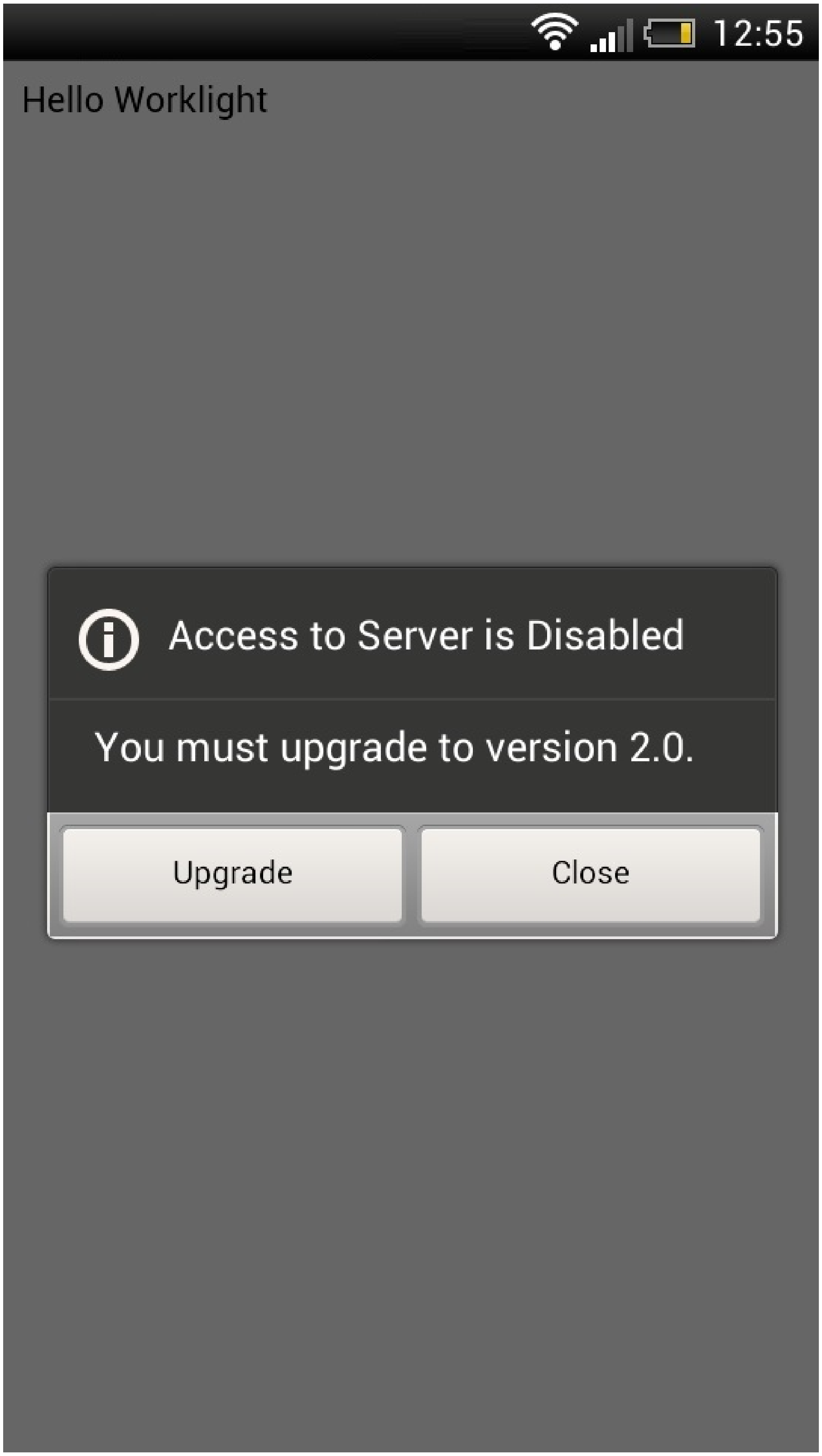
* Label: HelloWorld
Label of the application as defined by the developer

External URL: ibmappctr://show-app?id=com.HelloWorld
URL to open the Application Center mobile client on this application.

Figure 12-13. Copying the external URL from Application Details

3. When the external URL is copied to the clipboard, open the MobileFirst Operations Console.
4. Change the access rule of HelloWorld version 1.0 to "Access Disabled".
5. Paste the external URL into the URL field.

Running the client: When a mobile device connects to MobileFirst Server to try to run HelloWorld version 1.0, the device user is requested to upgrade the version of the application.



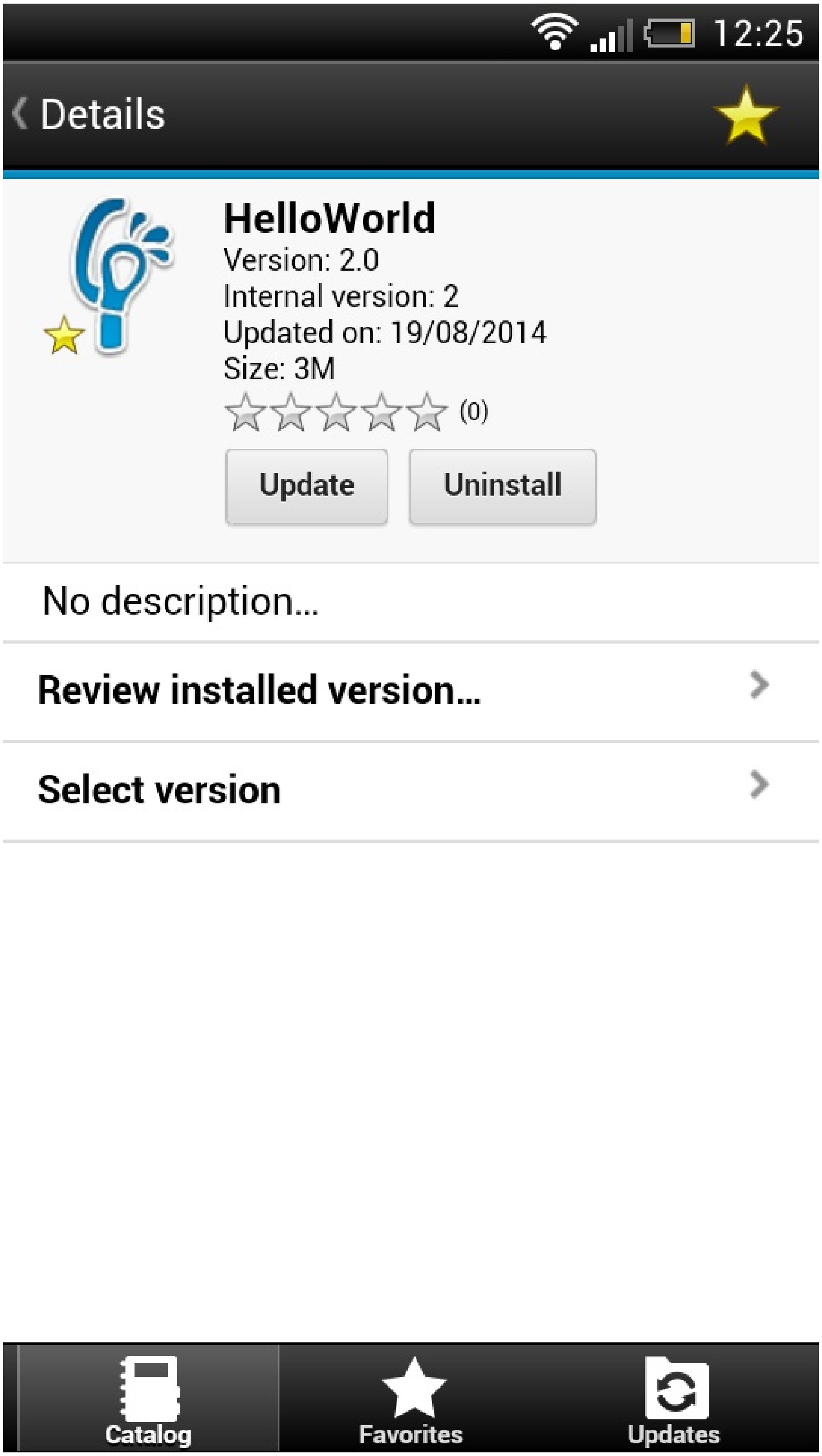
Hello Worklight

Access to Server is Disabled

You must upgrade to version 2.0.

Figure 12-14. 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.



HelloWorld
Version: 2.0
Internal version: 2
Updated on: 19/08/2014
Size: 3M

No description...

Figure 12-15. 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.

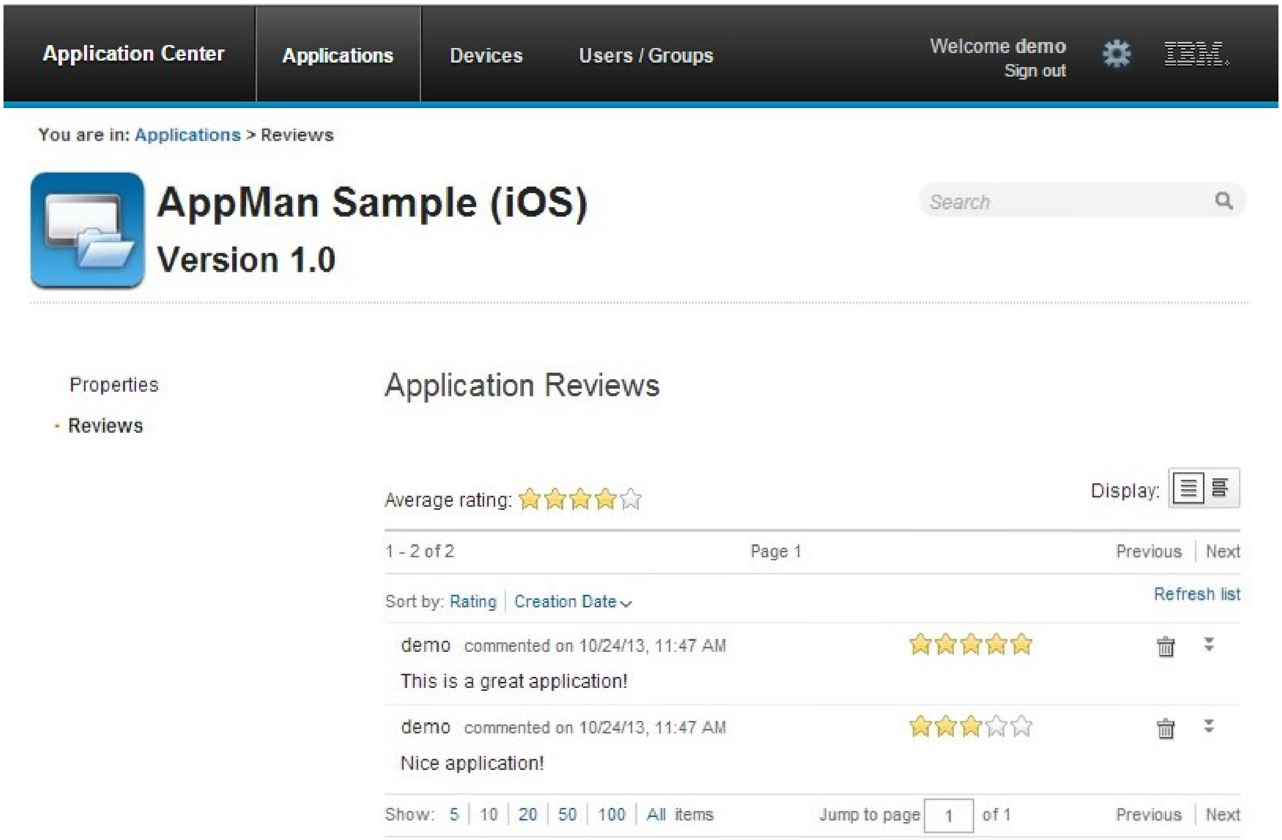
About this task

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**.



demo commented on 10/24/13, 11:47 AM ★★★★★

This is a great application!

demo commented on 10/24/13, 11:47 AM ★★★☆☆

Nice application!

Figure 12-16. Reviews of application versions

The rating is an average of the ratings in all recorded reviews. It consists of one to five stars, where one star represents the lowest level of appreciation and five stars represent the highest level of appreciation. The client cannot send a zero star rating.

The average rating gives an indication of how the application satisfies the intended use of the application.

4. Click the two arrow heads ⏬ on the right to expand the comment that is part of the review and to view the details of the mobile device where the review is generated.

   For example, the comment can give the reason for submitting the review, such as failure to install.

   If you want to delete the review, click the trash can on the right.

User and group management

You can use users and groups to define who has access to some features of the Application Center, such as installing applications on mobile devices.

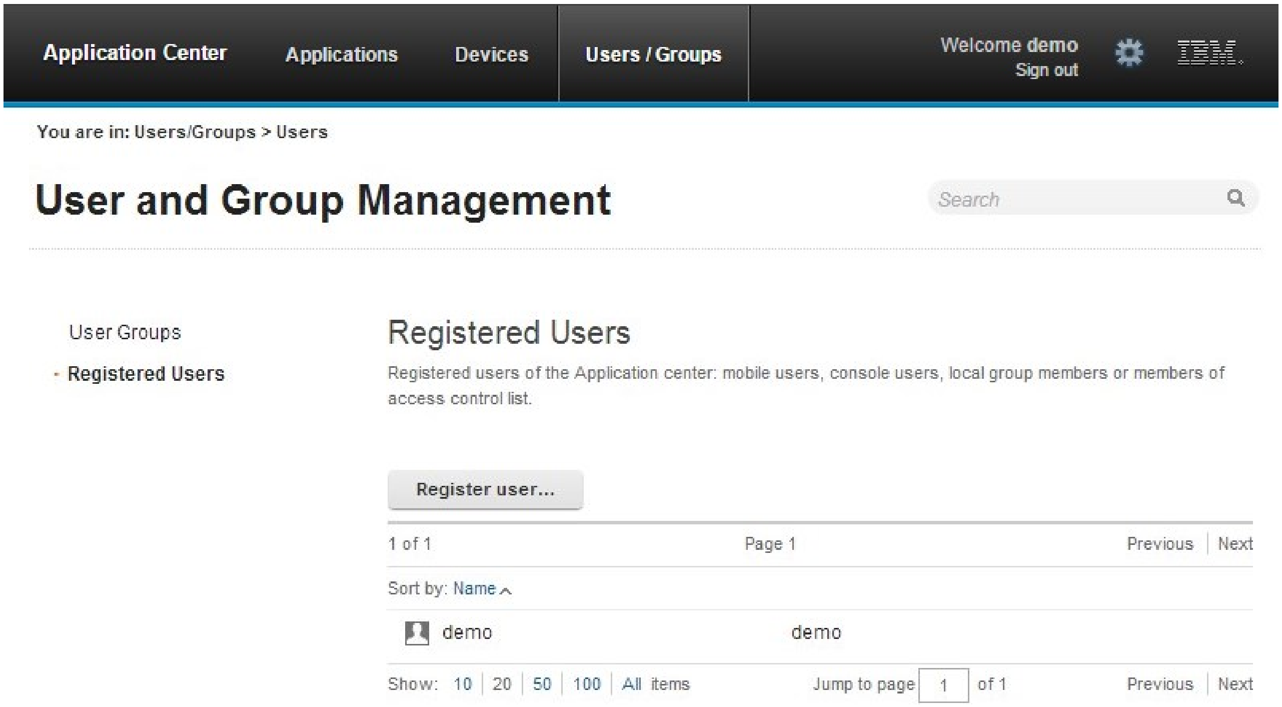
Purpose

Use users and groups in the definition of access control lists (ACL).

Managing registered users

To manage registered users, click the **Users/Groups** tab and select **Registered users**. You obtain a list of registered users of the Application Center that includes:

- Mobile client users
- Console users
- Local group members
- Members of an access control list



Registered users of the Application center: mobile users, console users, local group members or members of access control list.

Figure 12-17. 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:

XX

- Removing feedback given by the user
- Removing the user from the access control lists
- Removing the user from local groups

Note:

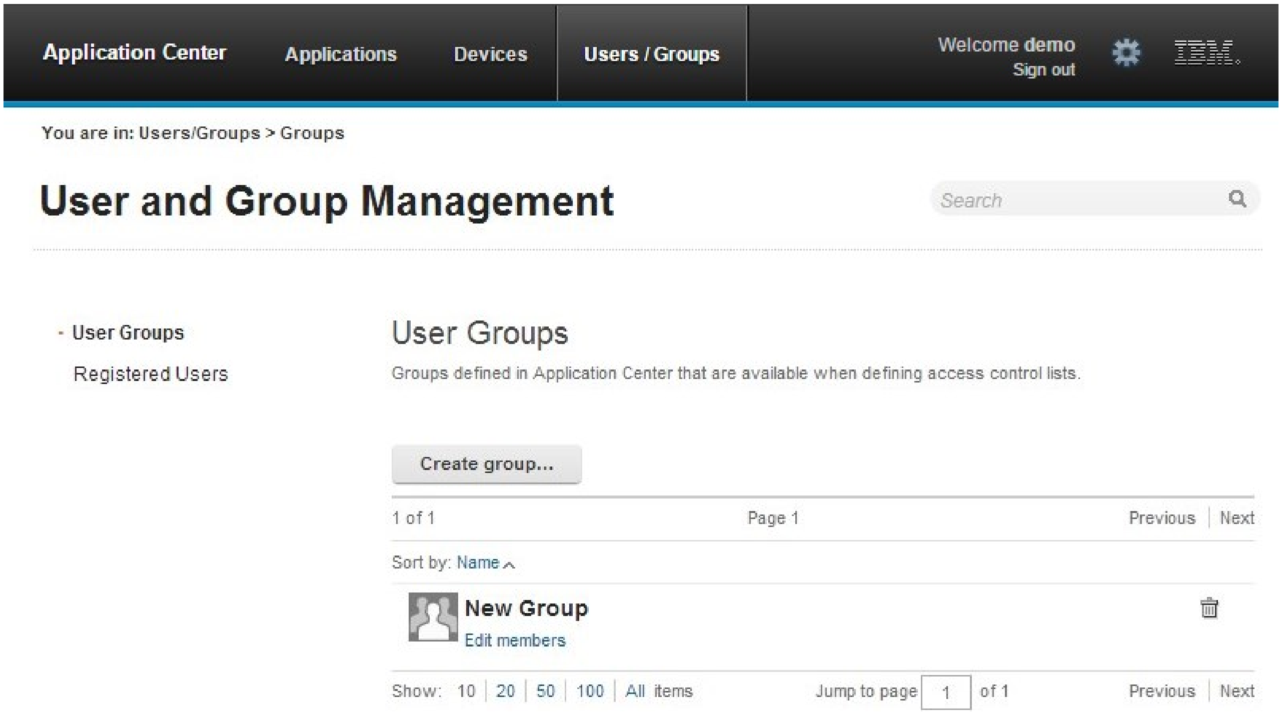
When you unregister a user, the user is not removed from the application server or the LDAP repository.

Managing local groups

To manage local groups, click the **Users/Groups** tab and select **User group**.

To create a local group, click **Create group**. Enter the name of the new group and click **OK**.

If the Application Center is connected to an LDAP repository, the search includes local groups as well as the groups defined in the LDAP repository. If the repository is not LDAP, only local groups are available to the search.



Groups defined in Application Center that are available when defining access control lists.

Figure 12-18. 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.

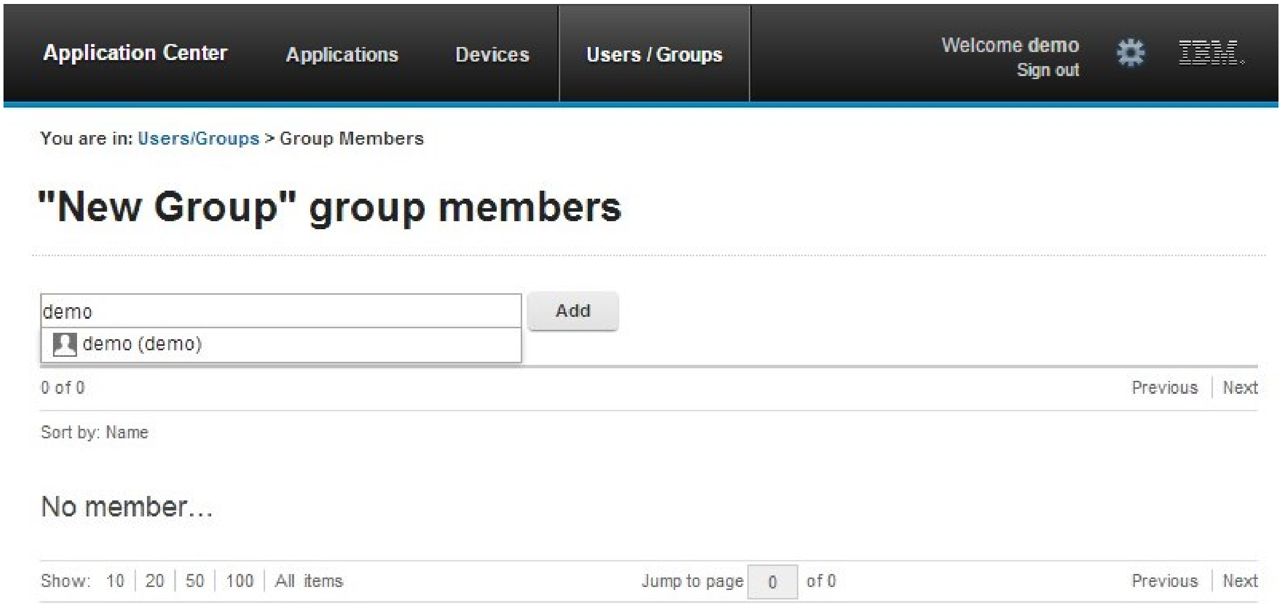


Figure 12-19. Managing group membership

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.

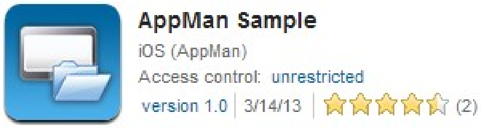
Managing access control

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

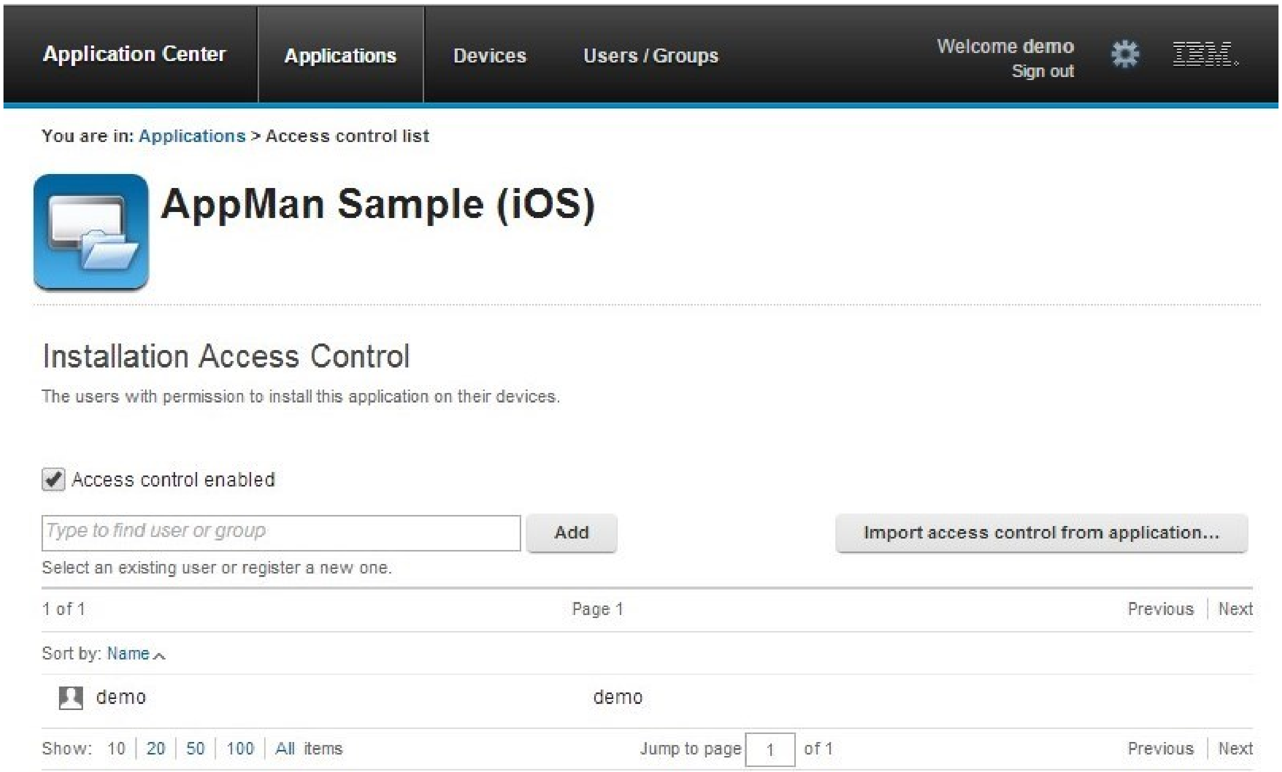
2. Select **Access control enabled** to enable access control.
3. Add users or groups to the access list.

   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.



You are in: Applications > Access control list

AppMan Sample (iOS)

Installation Access Control

The users with permission to install this application on their devices.

☑ Access control enabled

Type to find user or group

Select an existing user or register a new one.

1 of 1 Page 1

Sort by: Name

demo demo

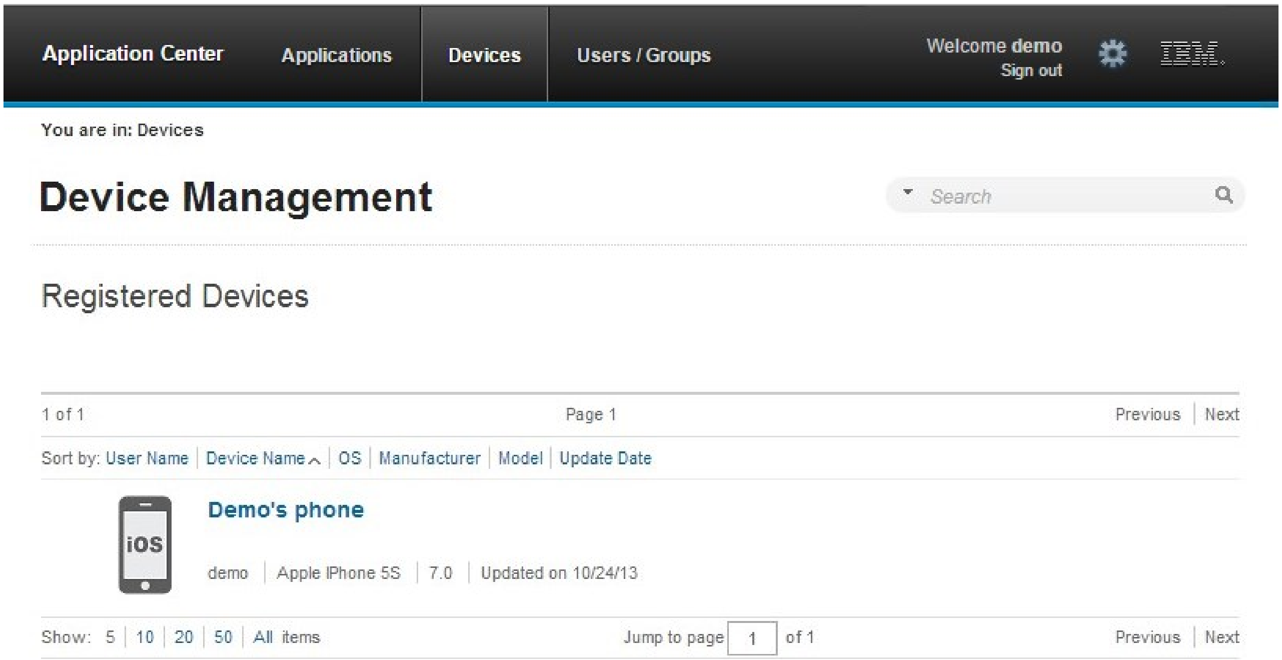
Figure 12-20. Adding users to the access list

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.



Device Management

Registered Devices

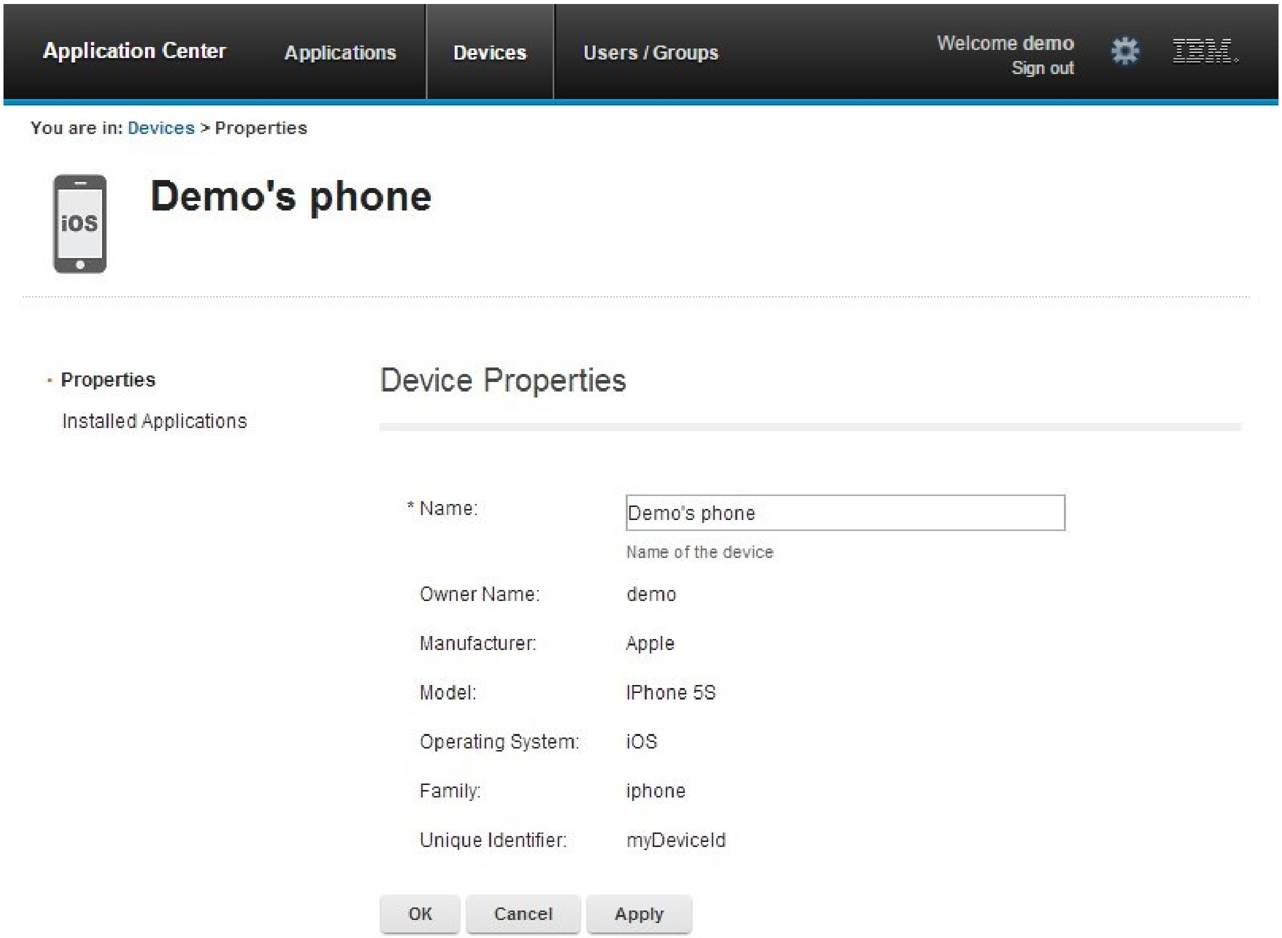
Demo's phone

demo | Apple IPhone 5S | 7.0 | Updated on 10/24/13

Figure 12-21. The device list

Device properties

Click a device in the list of devices to view the properties of the device or the applications installed on that device.



Demo's phone

Device Properties

* Name:	Demo's phone Name of the device
Owner Name:	demo
Manufacturer:	Apple
Model:	IPhone 5S
Operating System:	iOS
Family:	iphone
Unique Identifier:	myDeviceId

Figure 12-22. 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.

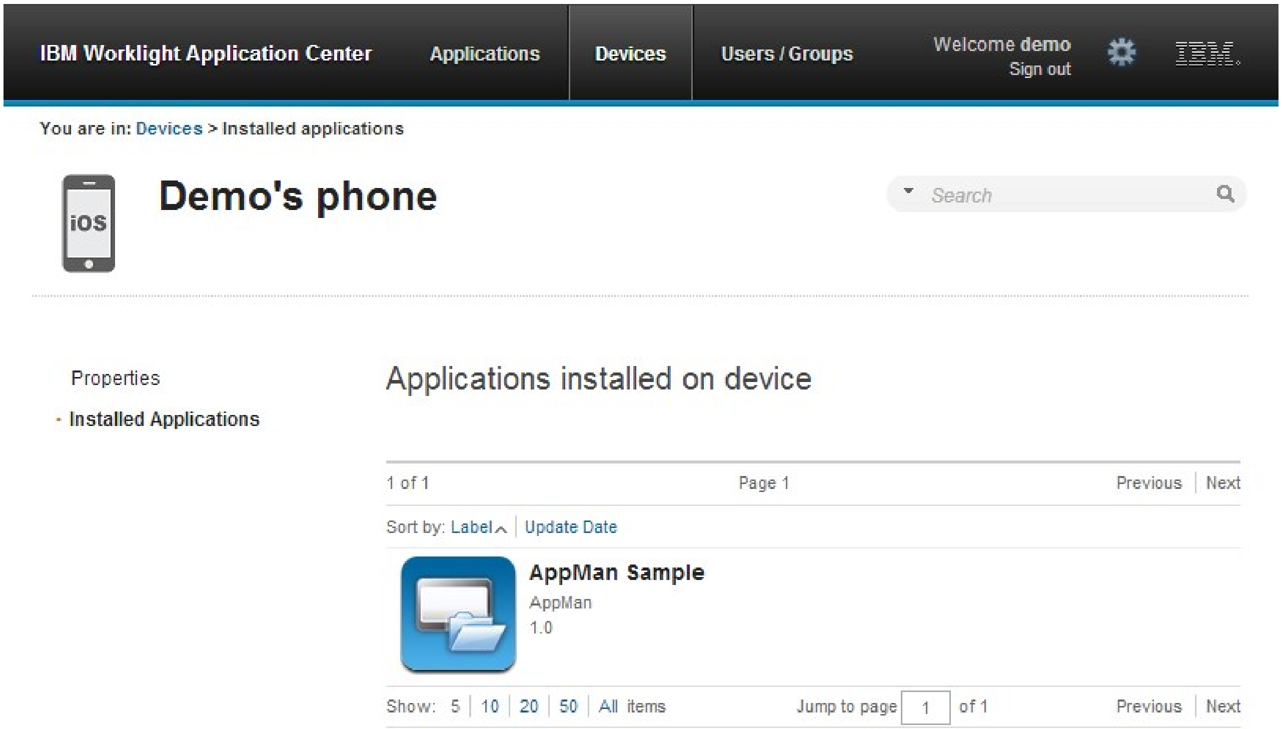


Figure 12-23. Applications installed on a device

Signing out of the Application Center console

For security purposes, you must sign out of the console when you have finished your administrative tasks.

Purpose

To log out of the secure sign-on to the Application Center console.

To sign out of the Application Center console, click **Sign out** next to the Welcome message that is displayed in the banner of every page.

Command-line tool for uploading or deleting an application

To deploy applications to the Application Center through a build process, use the command-line tool.

You can upload an application to the Application Center by using the web interface of the Application Center console. You can also upload a new application by using a command-line tool.

This is particularly useful when you want to incorporate the deployment of an application to the Application Center into a build process. This tool is located at:

installDir/ApplicationCenter/tools/applicationcenterdeploytool.jar

The tool can be used for application files with extension 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.

Option	Content indicated by	Description
`-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.
`-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 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 demopassword -f app1.ipa app2.ipa
```

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.

Option	Content indicated by	Description
-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.

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 iOS application demo.HelloWorld with internal version 3.0.

```
java com.ibm.appcenter.Upload -delete -s http://localhost:9080 -c applicationcenter -u demo -p
demopassword demo.HelloWorld@iOS@3.0
```

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
demo -p demopassword
```

Ant task for uploading or deleting an application

You can use the upload and delete tools as an Ant task and use the Ant task in your own Ant script.

Apache Ant is required to run these tasks. The minimum supported version of Apache Ant is listed in "System requirements" on page 2-9.

For convenience, Apache Ant 1.8.4 is included in IBM MobileFirst Platform Server. In the *product_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** value of the **upload** Ant task is com.ibm.appcenter.ant.UploadApps. The **classname** value 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.

Parameters of Ant task	Description
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.
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 available only in the **upload** Ant task.
file	The `.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 available only in the **delete** Ant task.
os	The operating system of the application. This parameter is available only in the **delete** Ant task.
version	The internal version of the application; this parameter is available only in the **delete** Ant task. Do not use the commercial version here, because the commercial version is unsuitable to identify the version exactly.

Example

You can find an extended example in the `ApplicationCenter/tools/build.xml` directory.

The following example shows how to use the Ant task in your own Ant script.

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="PureMeapAntDeployTask" basedir="." default="upload.AllApps">

  <property name="install.dir" value="../../" />
  <property name="workspace.root" value="../../" />

<!-- Server Properties -->
  <property name="server.path" value="http://localhost:9080/" />
  <property name="context.path" value="applicationcenter" />
  <property name="upload.file" value="" />
  <property name="force" value="true" />

  <!--  Authentication Properties -->
  <property name="login.user" value="appcenteradmin" />
  <property name="login.pass" value="admin" />
  <path id="classpath.run">
    <fileset dir="${install.dir}/ApplicationCenter/tools/">
      <include name="applicationcenterdeploytool.jar" />
      <include name="json4j.jar"/>
    </fileset>
  </path>
  <target name="upload.init">
    <taskdef name="uploadapps" classname="com.ibm.appcenter.ant.UploadApps">
      <classpath refid="classpath.run" />
    </taskdef>
  </target>
  <target name="upload.App" description="Uploads a single application" depends="upload.init">
    <uploadapps serverPath="${server.path}"
      context="${context.path}"
      loginUser="${login.user}"
      loginPass="${login.pass}"
      forceOverwrite="${force}"
      file="${upload.file}" />
    </target>
    <target name="upload.AllApps" description="Uploads all found APK and IPA files" depends="upload.init">
    <uploadapps serverPath="${server.path}"
      loginUser="${login.user}"
      loginPass="${login.pass}"
      forceOverwrite="${force}"
       context="${context.path}" >
      <fileset dir="${workspace.root}">
        <include name="**/*.ipa" />
      </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.ipa
```

You can also use it to upload all applications that are 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/
```

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 iOS device. 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. The user name and password are 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 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 6-275.

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 1 hour and does not require further 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 6-270.

Procedure

The installer is automatically started 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 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.

   If you open the page with HTTPS and use self-signed certificates, the browser displays a security warning about the SSL certificate, but lets you proceed to the website after confirmation that you consent to an unsafe connection.
3. Enter your user name and password.

   See Prerequisites in "The mobile client" on page 12-105.

   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.

   If you open the page with https:

   - If the web server uses a real SSL certificate that is provided by a trusted certificate authority, proceed to step 5 on page 12-107.

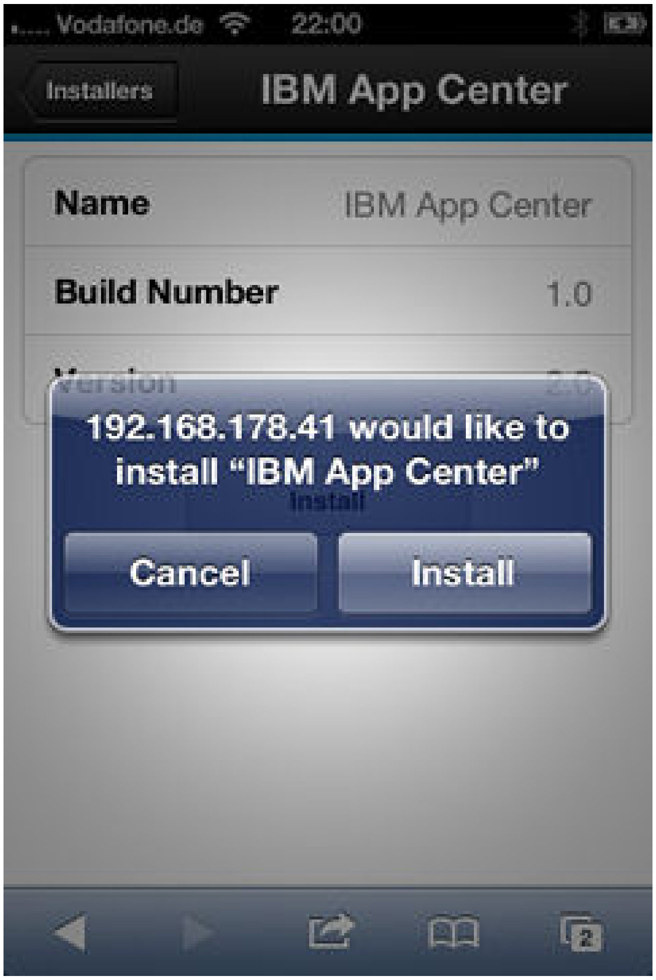
- If the web server uses a self-signed CA certificate, proceed to step 4.

4. If the web server uses a self-signed CA certificate, install the certificate at least once on the device.

   The Application Center administrator provides the certificate. See "Managing and installing self-signed CA certificates in an Application Center test environment" on page 6-278 for details.

   a. Tap the **SSL-Certificate** tab and select the certificate.
   b. Tap **Install**. You do this only once for the device. You can verify whether the certificate is installed by looking in **Settings** > **General** > **Profiles** on the device. This view shows the SSL certificates that the user installed on the device. If the self-signed CA certificate is not installed on the device, the iOS operating system prevents you from downloading the mobile client in the following steps.

      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 12-83.
5. Tap the **Installers** tab and select an item in the list to display the application details.
6. Tap **Install** to download the mobile client.
7. Enter your credentials to authorize the downloader transaction.
8. To authorize the download, tap **Install**.



192.168.178.41 would like to install "IBM App Center"

Figure 12-24. Confirm app to be installed

9. Enter your credentials to authorize the installation.
10. Close the browser.

    The app icon appears on the home screen and you can watch the download progress on the home screen.

Results

Note: 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, some versions of iOS 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.

The installation might be blocked for one of the following reasons:

- The provisioning profile of the application is not valid for the device. The application must be signed with a different provisioning profile.

- The device has no access to Apple servers to confirm the validity of the provisioning profile.
- The SSL certificate of the server is not known to the device.

What to do next

After the mobile client is installed on the device, you can open it.

In general, iOS applications can be installed on the device only if they are signed with a provisioning profile. See "Importing and building the project" on page 12-70.

Since iOS 9, when a company application is opened, depending on the type of the provisioning profile, an Untrusted Enterprise Developer message might display. This message explains that the provisioning profile is not yet trusted on this device. In this case, the application does not open, unless trust is established for this provisioning profile. Establishing trust must be done only once per provisioning profile.

To establish trust for a provisioning profile after the application is installed:

Until iOS 9.1

1. Go to **Settings** > **General** > **Profiles**.

   Under the **Enterprise apps** heading, you see the provisioning profile of the app.
2. Tap on the profile and confirm the trust.

Since iOS 9.2

1. Go to **Settings** > **General** > **Profiles** > **Device Management** or **Profiles & Device Management**.

   Under the **Enterprise apps** heading, you see the provisioning profile of the app.
2. Tap on the profile and confirm the trust.

After the trust is confirmed, no application that uses that provisioning profile shows the Untrusted Enterprise Developer message. For more information, see the Apple web site at `https://support.apple.com/en-us/HT204460`.

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.

On iOS devices, the credentials are saved in the keychain. After you successfully log in to the Application Center server, when you start the application subsequently, the login page is not displayed and the previous credentials are used. If the login cannot be performed, the login view is displayed.

SSL is mandatory on iOS devices. Therefore, this option is not displayed in the login view.

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 12-69 for information on filling parts of the address in advance, or hardcode the address and hide the associated fields.

Connecting to the server

To connect to the server:

1. Enter your user name and password.
2. Enter your Application Center server address.
3. 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 when you subsequently start 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.

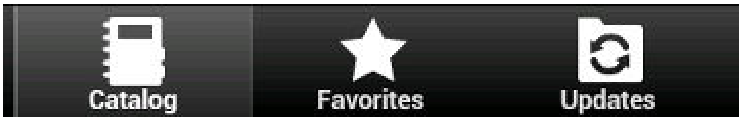


Figure 12-25. Views in the client application

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.

Updates

This view shows all applications that you marked as favorite apps and that have a later version available in 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

Different device types might have quite different page displays. On the phone, a list is displayed. On a tablet, a grid of applications is used.

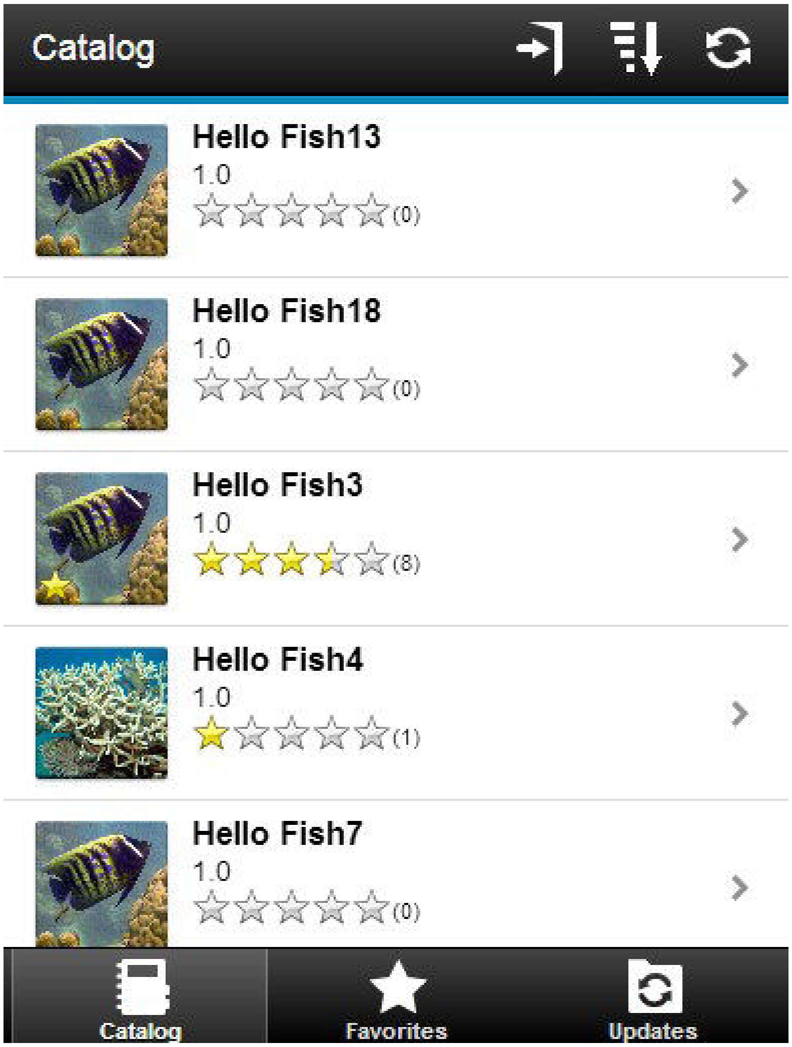


Figure 12-26. Catalog view on a phone

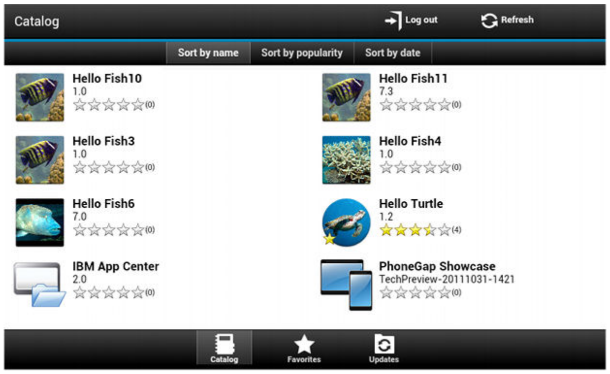


Figure 12-27. Catalog view on a tablet

Features of the views

On a tablet, you can sort the lists by tapping one of the sort criteria.

Sort criteria are available through the sort button.

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 12-69 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:

- Tap the logout button. ⇥

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 iOS, the build number of the application; See "Application properties" on page 12-83 for technical details concerning this property.
- Update date.
- Approximate size of the application file.
- Rating of the version and number of ratings received.
- Description of the application.

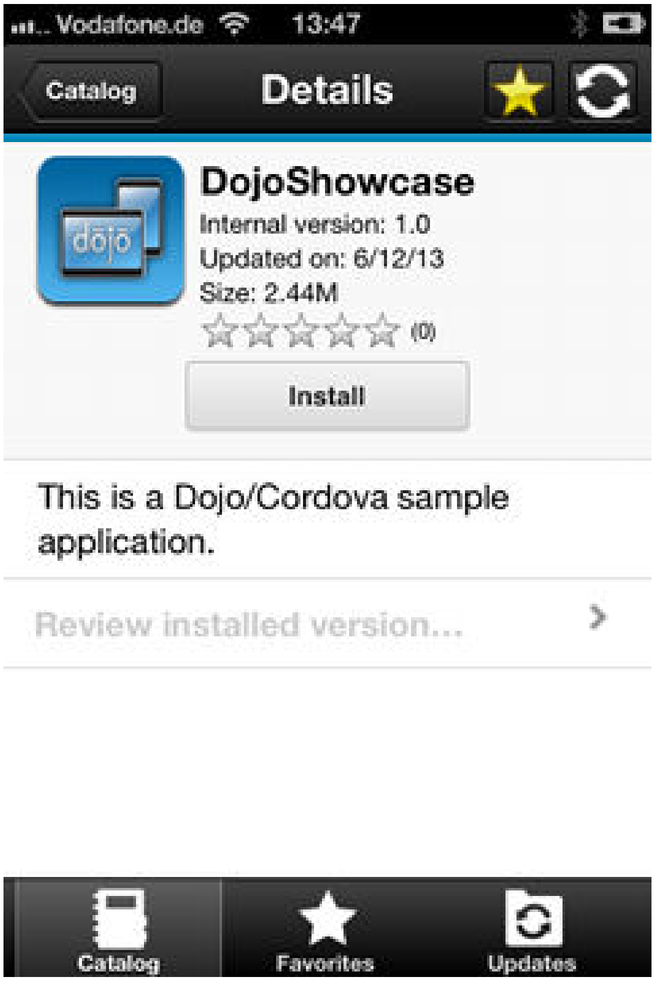
You can perform several actions in this view.

- Install, upgrade, downgrade, or uninstall an application version.
- Cancel the current operation in progress (if available).
- 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 iOS device

From the **Details** view, you can install an application version on your iOS mobile device.

About this task



DojoShowcase

Internal version: 1.0

Updated on: 6/12/13

Size: 2.44M

☆☆☆☆☆ (0)

This is a Dojo/Cordova sample application.

Review installed version...

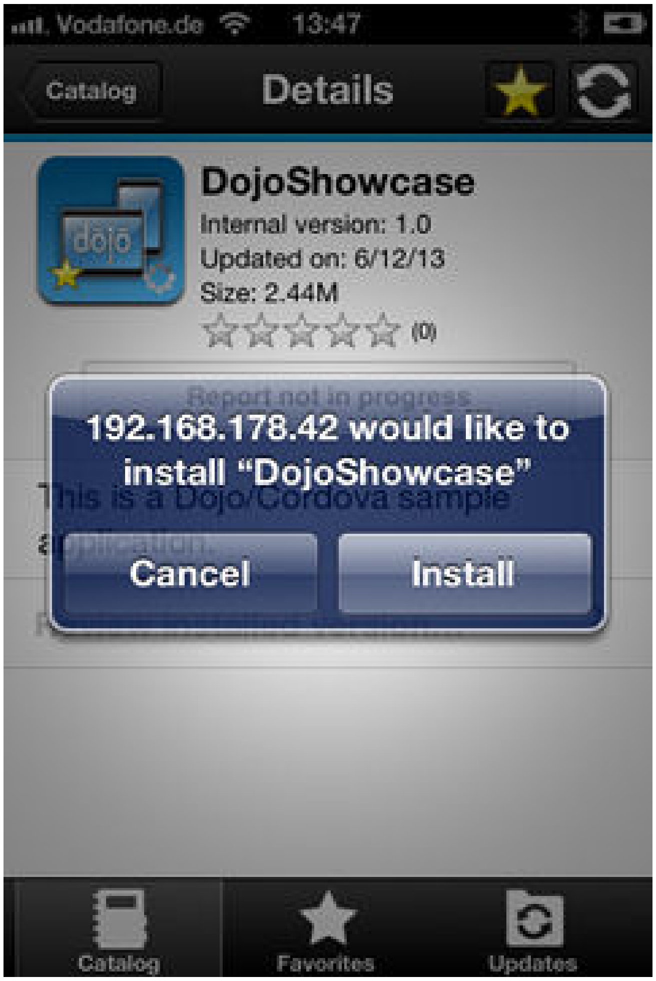
Figure 12-28. Details view of an app version shown on your iOS device

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 6-275.

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.



DojoShowcase
Internal version: 1.0
Updated on: 6/12/13
Size: 2.44M
(0)

192.168.178.42 would like to install “DojoShowcase”

Figure 12-29. Canceling application installation on your iOS device

Depending on the action that is 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 6 (deprecated) or earlier gives an error message.

Results

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.

What to do next

After the application is installed on the device, you can open it.

In general, iOS applications can be installed on the device only if they are signed with a provisioning profile. See "Importing and building the project" on page 12-70.

Since iOS 9, when a company application is opened, depending on the type of the provisioning profile, an Untrusted Enterprise Developer message might display. This message explains that the provisioning profile is not yet trusted on this device. In this case, the application does not open, unless trust is established for this provisioning profile. Establishing trust must be done only once per provisioning profile.

To establish trust for a provisioning profile after the application is installed:

Until iOS 9.1

1. Go to **Settings** > **General** > **Profiles**.

   Under the **Enterprise apps** heading, you see the provisioning profile of the app.
2. Tap on the profile and confirm the trust.

Since iOS 9.2

1. Go to **Settings** > **General** > **Profiles** > **Device Management** or **Profiles & Device Management**.

   Under the **Enterprise apps** heading, you see the provisioning profile of the app.
2. Tap on the profile and confirm the trust.

After the trust is confirmed, no application that uses that provisioning profile shows the Untrusted Enterprise Developer message. For more information, see the Apple web site at `https://support.apple.com/en-us/HT204460`.

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.

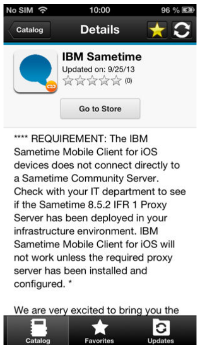
About this task

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 12-81. You can install these applications through the mobile client on your compatible device.

Links 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 Apple iTunes.



**** REQUIREMENT: The IBM Sametime Mobile Client for iOS devices does not connect directly to a Sametime Community Server. Check with your IT department to see if the Sametime 8.5.2 IFR 1 Proxy Server has been deployed in your infrastructure environment. IBM Sametime Mobile Client for iOS will not work unless the required proxy server has been installed and configured. *

We are very excited to bring you the

Figure 12-30. 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.

About this task

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.

Showing details of a specific application version

Select a version of an application to show its details.

Procedure

1. Show details of a specific application version on a mobile device by selecting the appropriate procedure to follow for your device.
   - A phone; see step 2.
   - A tablet; see step 3 on page 12-117.
2. Show details of a specific application version on an iOS phone.
   a. Tap **Select a version** to navigate to the version list view.

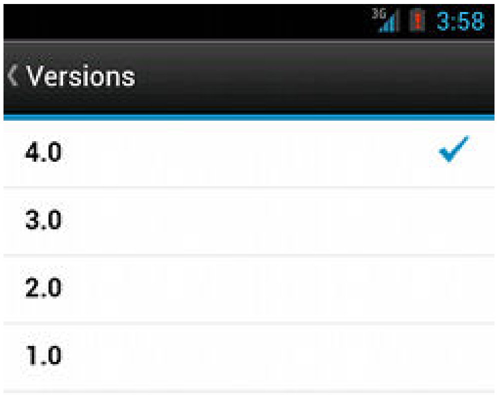


Figure 12-31. Specific version of an application selected in the list of versions on an iOS phone

   b. Tap the required version of the application. The **Details** view is updated and shows the details of the selected application version.

3. **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 more recent version of the application or take the latest available version.
3. On iOS devices, tap **Install latest**.
4. Follow the appropriate application installation procedure. "Installing an application on an iOS device" on page 12-113.

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.

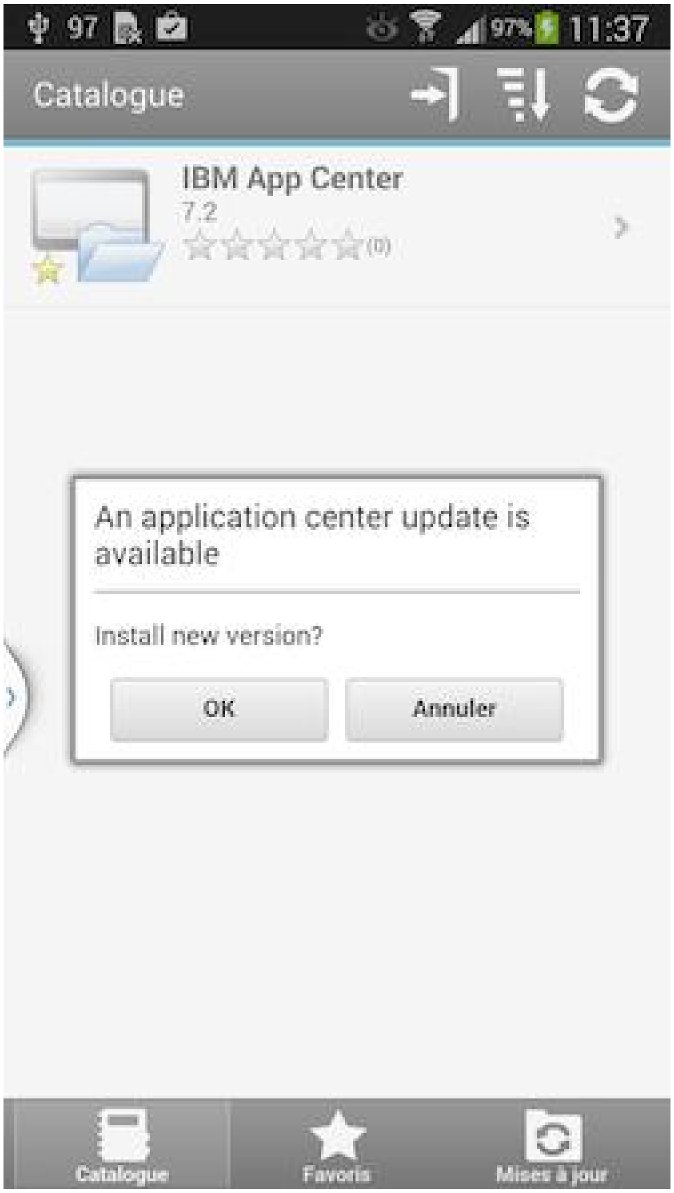
If a later version of the application is detected, you are requested to download and install the later version.

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 MobileFirst 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.

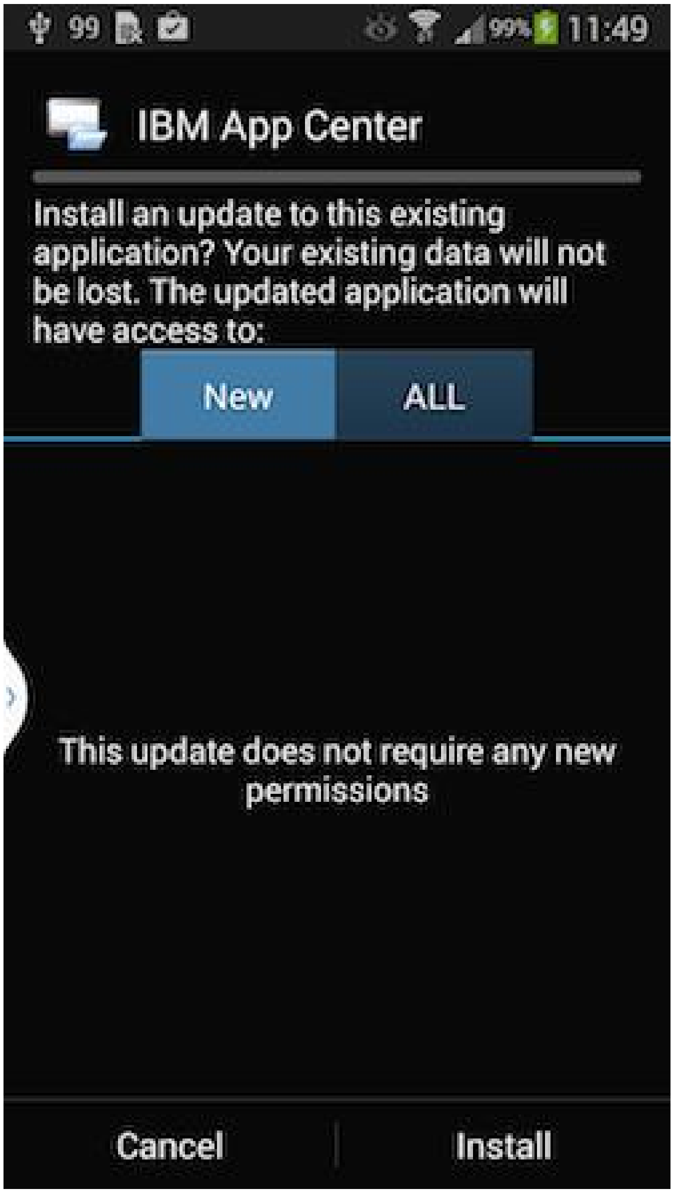


An application center update is available

Install new version?

Figure 12-32. Detection of a later version of the client application available on the server

2. Tap **Install** to install the later version of the application.



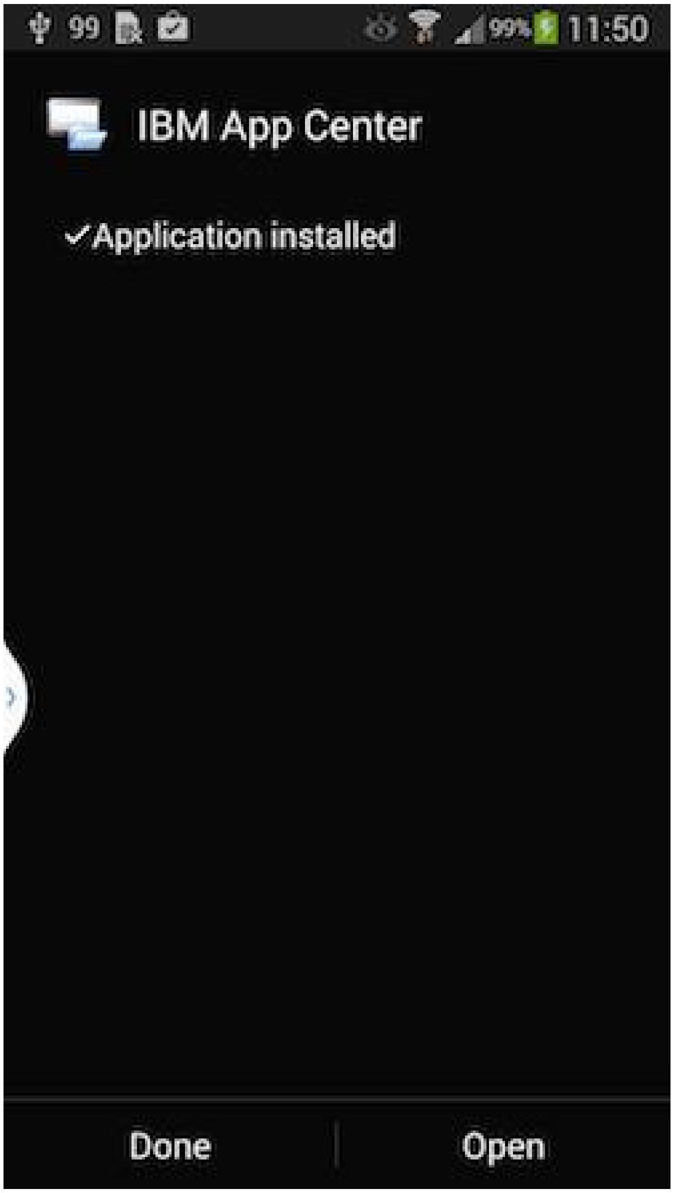
IBM App Center

Install an update to this existing application? Your existing data will not be lost. The updated application will have access to:

This update does not require any new permissions

Figure 12-33. Confirm installation of the updated version of the application

3. Tap **Open** to start the updated application.

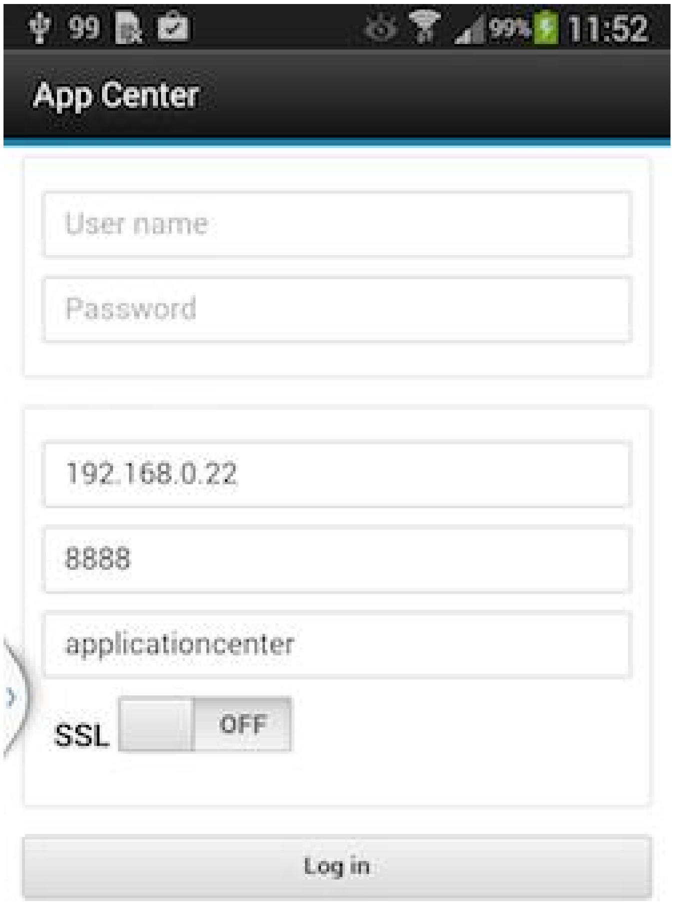


✓Application installed

Figure 12-34. Starting the updated application

Results

You must log in to the updated version of the application to run it.



User name

Password

192.168.0.22

8888

applicationcenter

SSL OFF

Figure 12-35. Logging in to the new version of the client application

Note: To upgrade the Application Center client, the following conditions apply:

- The new Application Center client must use the same package name or bundle identifier as the old client.
- The new Application Center client must be signed with the same provisioning profile as the old client.

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 12-116 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 12-69 for information about how to disable reverting to earlier versions of an application.

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 12-113.

Marking or unmarking a favorite app

Mark your favorite apps or unmark an app to have it removed from the favorites list.

An application that is 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 iOS phones and tablets, tap **Review version X**.
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.

   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.

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 phone
Tap **View Reviews** to navigate to the **Reviews** view.

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 phone
Tap **View Reviews** to navigate to the **Reviews** view. Then tap the settings icon ⚙, tap **All versions**, and confirm the selection.

On a tablet
Tap **All Reviews**.

Federal standards support in IBM MobileFirst Platform Foundation for iOS

IBM MobileFirst Platform Foundation for iOS supports Federal Desktop Core Configuration (FDCC), and United States Government Configuration Baseline (USGCB) specifications. IBM MobileFirst Platform Foundation for iOS 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, 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 MobileFirst Server, and SSL communications with the MobileFirst Server

The IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Platform Foundation for iOS client transacts a Secure Socket Layer (SSL) connection to a MobileFirst 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. 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 12-127 for links to documentation to enable FIPS 140-2 mode in WebSphere Application Server.

FIPS 140-2 on the MobileFirst 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 MobileFirst Platform Foundation for iOS. Protection of data in motion is provided by the use of HTTPS communication between the MobileFirst client and the MobileFirst Server. By default, these features use non-FIPS 140-2 validated libraries. But for iOS 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 MobileFirst Server. This support is achieved by using an OpenSSL library that achieved FIPS 140-2 validation (Certificate #1747). To enable this option in a MobileFirst client project, select the FIPS 140-2 optional feature in the MobileFirst 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 MobileFirst client and the MobileFirst Server.
- This feature is supported on **i386**, **armv7**, and **armv7s** architectures.
- It only works with hybrid applications (not native).
- For HTTPS communications:
  - Only the communications between the MobileFirst client and the MobileFirst 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 MobileFirst client can only communicate with a MobileFirst Server that runs in supported environments, which are listed in the System Requirements. If the MobileFirst 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.
- IBM MobileFirst Platform 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>
  ```

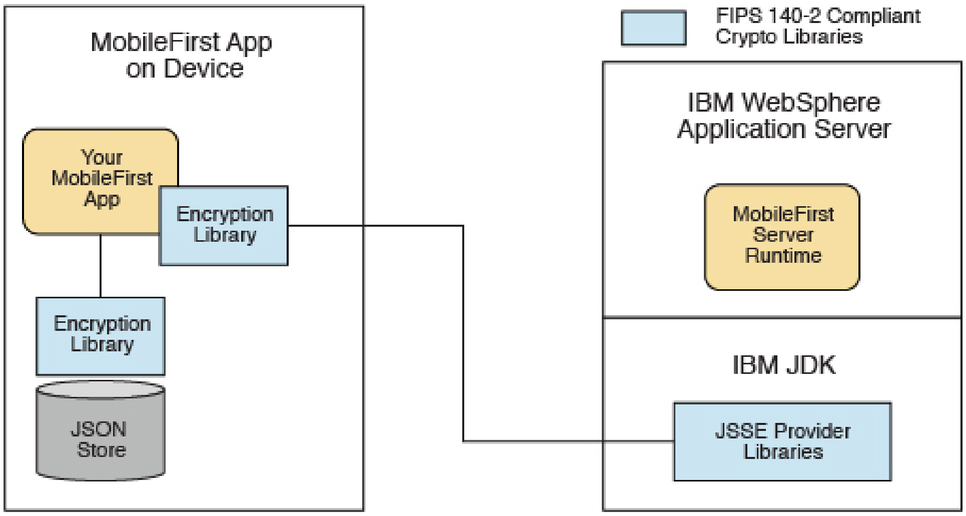
  Use this mobile security test for iOS environments as follows:

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

The FIPS 140-2 optional feature supersedes the changes that were described in tutorial *JSONStore - Encrypting sensitive data with FIPS 140-2* that was available for IBM MobileFirst Platform Foundation for iOS V6.0.0 and earlier, *Advanced client side development*. This tutorial is available in the Advanced client side development category on the Getting Started page of the Developer Center.

If you previously made the changes that are described in the tutorial, you must first save any other environment-specific changes that you made, and then delete and re-create your iOS environment.

Figure 12-36. Example


FIPS 140-2 Compliant Crypto Libraries
MobileFirst App on Device
Your MobileFirst App
Encryption Library
Encryption Library
JSON Store
IBM WebSphere Application Server
MobileFirst Server Runtime
IBM JDK
JSSE Provider Libraries

For more information about JSONStore, see "JSONStore overview" on page 8-125.

References

For information about how to enable FIPS 140-2 mode in WebSphere Application Server, see Federal Information Processing Standard support.

For the WebSphere Application Server Liberty profile, no option is available in the administrative console to enable FIPS 140-2 mode. But you can enable FIPS 140-2 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.

Monitoring and mobile operations

IBM MobileFirst Platform Foundation for iOS includes a range of operational analytics and reporting mechanisms for collecting, viewing, and analyzing data from your IBM MobileFirst Platform Foundation for iOS applications and servers, and for monitoring server health.

Logging and monitoring mechanisms

IBM MobileFirst Platform Foundation for iOS reports errors, warnings, and informational messages into a log file. The underlying logging mechanism varies by application server.

IBM MobileFirst Platform Server

MobileFirst Server uses the standard java.util.logging package. By default, all MobileFirst logging goes into the application server log files. You can control MobileFirst 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 MobileFirst logs. MobileFirst logs all begin with "com.worklight".

Application Center logs begin with “com.ibm.puremeap”.

For information on sending logging information from packages outside of the MobileFirst package (`com.worklight`), see Adding additional logger packages.

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 13-1. 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	https://www.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/rwlp_logging.html

Log level mappings

MobileFirst 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 MobileFirst 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 MobileFirst 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

MobileFirst Operations 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 MobileFirst Administration service (`worklightadmin.war`) to false.

When the audit log is enabled, you can download it from MobileFirst Operations 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 MobileFirst vitality queries to run a health check of your server, and determine the vitality status of your server.

You generally use the MobileFirst 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 13-2. Examples of queries that help determine server vitality

Query	Purpose
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality`	Checks the server as a whole.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp`	Checks the server and the `MyApp` application.
`http://<server>:<port>/<publicWorkLightContext>/ws/rest/vitality?app=MyApp&adapter=MyAdapter`	Checks the server, the `MyApp` application, and the `MyAdapter` adapter.

Note: Do not include the `/<publicWorkLightContext>` part of the URL if you use IBM MobileFirst Platform Foundation Developer Edition. You must add this part of the URL only if MobileFirst 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 13-3. Return values and values

Return attribute	Possible values
DATE	Date value in JavaScript™ format
EVENTID	0 for the running server, deployed adapter, or deployed application 1 for not deployed adapter 2 for not deployed application 3 for malfunctioning server
SUBJECT	SRV for MobileFirst Server ADPT for adapter APPL for application BUILD for the version of the MobileFirst 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.

Setting logging and tracing for Application Center on the application server

You can set logging and trace parameters for particular application servers and use JNDI properties to control output on all supported application servers.

You can set the logging levels and the output file for tracing operations for Application Center in ways that are specific to particular application servers. In addition, IBM MobileFirst Platform Foundation for iOS provides Java Naming and Directory Interface (JNDI) properties to control the formatting and redirection of trace output, and to print generated SQL statements.

Enabling logging and tracing in WebSphere Application Server full profile

You can set the logging levels and the output file for tracing operations on the application server.

About this task

When you try to diagnose problems in the Application Center (or other components of IBM MobileFirst Platform Foundation for iOS), it is important to be able to see the log messages. To print readable log messages in log files, you must specify the applicable settings as Java virtual machine (JVM) properties.

Procedure

1. Open the administrative console of WebSphere Application Server.
2. Select **Troubleshooting** > **Logs and Trace**.

3. In “Logging and tracing”, select the appropriate application server and then select “Change log detail levels”.
4. Select the required packages and their corresponding detail level. For example, in this way you can select following packages and set the detail level. This example enables logging for IBM MobileFirst Platform Foundation for iOS, including Application Center, with level FINEST (equivalent to ALL)

   ```
   com.ibm.puremeap.*=all
   com.ibm.worklight.*=all
   com.worklight.*=all
   ```

   Where:

   - com.ibm.puremeap.* is for Application Center.
   - com.ibm.worklight.* and com.worklight.* are for other MobileFirst components.

   The traces are sent to a file called trace.log. Note that they are not sent to SystemOut.log or to SystemErr.log.

   For more details, see Configuring Java logging using the administrative console.

Enabling logging and tracing in WebSphere Application Server Liberty profile

You can set the logging levels and the output file for tracing operations for Application Center on the Liberty profile application server.

When you try to diagnose problems in the Application Center, it is important to be able to see the log messages. To print readable log messages in log files, you must specify the applicable settings.

To enable logging for IBM MobileFirst Platform Foundation for iOS, including Application Center, with level FINEST (equivalent to ALL), add a line to the server.xml file. For example:

```
<logging traceSpecification="com.ibm.puremeap.*=all:com.ibm.worklight.*=all:com.worklight.*=all"/>
```

Where multiple entries of a package and its logging level are separated by a colon (:).

The traces are sent to a file called trace.log. Note that they are not sent to messages.log or to console.log.

For more details, see Liberty profile: Logging and Trace.

Enabling logging and tracing in Apache Tomcat

You can set the logging levels and the output file for tracing operations undertaken on the Apache Tomcat application server.

When you try to diagnose problems in the Application Center, it is important to be able to see the log messages. To print readable log messages in log files, you must specify the applicable settings.

To enable logging for IBM MobileFirst Platform Foundation for iOS, including Application Center, with level FINEST (equivalent to ALL), edit the conf/logging.properties file. For example, add lines similar to these lines:

```
com.ibm.puremeap.level = ALL
com.ibm.worklight.level = ALL
com.worklight.level = ALL
```

For more details, see Logging in Tomcat.

JNDI properties for controlling trace output

On all supported platforms, you can use Java Naming and Directory Interface (JNDI) properties to format and redirect trace output for Application Center, and to print generated SQL statements.

The following table shows the applicable properties and settings.

Table 13-4. JNDI property settings for controlling trace output

Property settings	Description
ibm.appcenter.logging.formatjson=true	This setting uses white space to format the JSON output for easier reading in log files.
ibm.appcenter.logging.tosystemerror=true	This setting prints all log messages to system error in log files. It enables you to turn on logging globally.
ibm.appcenter.openjpa.Log=DefaultLevel=WARN, Runtime=INFO, Tool=INFO, SQL=TRACE	This setting prints all the generated SQL statements in the log files.

Operational analytics

The operational analytics feature enables searching across apps, services, devices, and other sources to collect data about usage or detect problems. The operational analytics platform collects data about applications, adapters, devices, and logs to give a high-level view of the client interaction with the MobileFirst Server and to enable problem detection.

IBM MobileFirst Platform Foundation for iOS includes a scalable operational analytics feature that is accessible from the MobileFirst Operations 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 MobileFirst client/server protocol, including push notification).
- Client-side logs and crashes.
- Server-side logs that are captured in traditional MobileFirst log files.

The data for operational analytics includes the following sources:

- Crash events of an application on iOS devices (crash events for native code errors).
- Interactions of any application-to-server activity (anything that is supported by the MobileFirst client/server protocol, including push notification).
- Server-side logs that are captured in traditional MobileFirst log files.

The operational analytics feature is accessible from the MobileFirst Operations Console and includes the following capabilities:

- Near real-time analytics for client activity with the MobileFirst Server.
- Analytics for adapter hits.

- Network latency analytics.
- Client log search and download.
- Server log search and download.
- Crash and stack trace search.

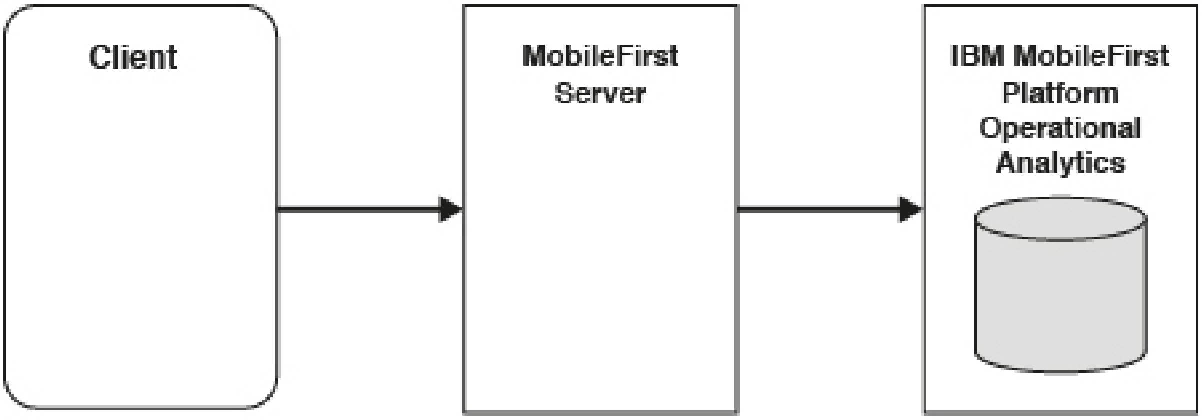
All data collected by the analytics platform can be visualized through the Operational Analytics console. The console also provides the ability to create custom charts based on data collected by the analytics platform. In addition to an at-a-glance view of your mobile and web application analytics, the 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 MobileFirst Platform Operational Analytics.

Data capture

When the IBM MobileFirst Platform Operational Analytics is deployed and the MobileFirst Server is properly configured, data begins to flow from the MobileFirst Server to the IBM MobileFirst Platform Operational Analytics. 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 MobileFirst Server.

Analytics event types

The following image shows the analytics data flow:



All data that is sent from the MobileFirst Server to the IBM MobileFirst Platform Operational Analytics is categorized by its type. This section briefly describes the different types of analytics data that is captured and analyzed.

App Activities

All IBM MobileFirst Platform Foundation for iOS client/server network communication is considered to be an app activity. An app activity is sent to the IBM MobileFirst Platform Operational Analytics when:

- A client device begins a new session with the MobileFirst Server.
- A client device makes an adapter request.

When the client communicates with the MobileFirst Server through one of the previously mentioned events, it also sends metadata about the device, including:

- Device environment (iOS, and so on).
- Device model (iPhone3,3, iPad4,4, and so on).

  Note: MobileFirst Operational Analytics logs the iPhone model number. On iOS, two different values are reported for the device model: the generation and the hardware identifier string. The hardware identifier string comes from Cordova, while the generation comes from the app instance registration. Due to this difference, you might see some iOS device models in the Analytics Console being referred to by either string.
- 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 MobileFirst Server activity produces log messages that are saved to the disk. These messages are also forwarded to theMobileFirst Server IBM MobileFirst Platform Operational Analytics and can be searched.

Client Logs
Client devices can be configured to capture log data and crash events to be forwarded to the MobileFirst Server. For more information, see “Manually captured data.”

Notification Activities
Upon a successful push notification, a notification activity is automatically sent to the IBM MobileFirst Platform Operational Analytics.

Manually captured data

The following data must be captured manually by changing the client application.

Client Logs
The following example shows how to create client logs to be sent to the IBM MobileFirst Platform Operational Analytics.

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 MobileFirst Server
[OCLogger send];
```

For more information about capturing client-side logs, see “Client-side log capture configuration from MobileFirst Operations Console” on page 8-311.

All persisted client-side logs are sent automatically upon a successful initialization (successful session start) with the MobileFirst 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 MobileFirst Server (successful session start) and when the client calls the `send()` method.

Collecting Analytics for Network Connections to other servers

All network communications to the MobileFirst Server are logged and collected for analytics as part of MobileFirst Operational Analytics. To enable collection of network analytics to servers other than the MobileFirst Server, use `WLResourceRequest` provided in IBM MobileFirst Platform Foundation for iOS.

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 );
WL.Analytics.send();
```

For example:

```
WL.Analytics.log( { "hello": "world" } , "This is an analytics log" );
WL.Analytics.send();
```

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 to FATAL. However, logs that are produced by `WL.Analytics.log` are always captured, despite the current logging level. These logs are sent to the Analytics Server only after `WL.Analytics.send()` is called.

The metadata object that is passed into this method is searchable. However, the Analytics Console does not make further use of the object beyond the custom activities that are shown next. The primary purpose of the metadata object is to allow you to log custom JSON objects if you want to export analytics data into their own custom tool.

Crash reports

If you want uncaught exceptions recorded and sent to IBM MobileFirst Platform Operational Analytics 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: %@. Reason: %@", e.name, e.reason];
}
```

Note that the logs are not sent to the server until the application is restarted and connects to the MobileFirst Server.

Custom Activities

The 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 MobileFirst Platform Operational Analytics that can be searched on the Activities page of the AnalyticsConsole.

Client configurations

No additional client configurations are needed for client devices to forward analytics data to the MobileFirst Server.

The Analytics Optional Feature is not required to be enabled on the client for analytics in IBM MobileFirst Platform Foundation for iOS V7.1.0.

The `analyticsEnabled` flag in the `initOptions.js` file is not required to be enabled on the client for analytics in IBM MobileFirst Platform Foundation for iOS V7.1.0.

These configurations are only necessary if you are using the previous analytics platform (IBM SmartCloud Analytics Embedded). If you are using the new IBM MobileFirst Platform Operational Analytics in IBM MobileFirst Platform Foundation for iOS V7.1.0, then these properties can be ignored.

Event types

Data that is sent to the MobileFirst Operational Analytics is categorized into an event type.

MobileFirst Operational Analytics contains the following event types:

- Custom Data
- Devices
- Device App Logs
- Device App Network Transactions
- Device App Push Action
- If you are using the latest interim fix of MobileFirst: “Device App Session” on page 13-14
- Mobile Users
- Server Logs
- Server Network Transactions
- Server Push Notifications
- Server Push Subscriptions

Custom Data

Table 13-5. Custom Data Properties.

Property Name	Description
`appID`	Unique Bluemix application identifier
`clientID`	Unique client identifier
`deviceAppName`	Name of the MobileFirst application
`deviceAppVersion`	Version of the MobileFirst application
`deviceID`	Unique device identifier
`deviceModel`	Device model name. For example: iPhone3,3
`deviceOS`	Device environment. For example: iPhone

Table 13-5. Custom Data Properties (continued).

Property Name	Description
deviceOSversion	Device operating system version. For example: 7.0
hostname	Host name of the MobileFirst Analytics server
serverIpAddress	IP address of the MobileFirst Analytics server
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes

In addition to the default properties that are logged, any custom events that are sent to MobileFirst Operational Analytics by the `WL.Analytics.log` method also become a property of the custom data event.

Devices

Table 13-6. Devices Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
firstAccessTimestamp	Time stamp of the device's first access to a specific version of a MobileFirst application

Device App Logs

Table 13-7. Device App Logs Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0

Table 13-7. Device App Logs Properties (continued).

Property Name	Description
file	File name of the code that recorded this log entry
level	Level at which the log was recorded. For example: WARN, DEBUG
line	Line number of the code that recorded this log entry
pkg	Name of the logger instance that recorded this log entry
method	Name of the method that recorded this log entry
message	Message content of the log
stacktrace	Stack trace that is associated with the log
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes
userID	Unique user identifier
wifiAPS	Wireless access points
wifiConnectedMAC	MAC address of the connected wireless network
wifiConnectedSSID	Name of the connected wireless network
wifiTimestamp	Last recorded time that the client connected to the wireless network

Device App Network Transactions

Table 13-8. Device App Network Transactions Properties.

Property Name	Description
appID	Unique Bluemix application identifier
bytesReceived	Number of bytes that were received
bytesSent	Number of bytes that were sent
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceIpAddress	IP address of the device
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
inboundTimestamp	Time when the client receives a response to a network request
networkType	Type of network connection that was used. For example: Wi-Fi, 3G

Table 13-8. Device App Network Transactions Properties (continued).

Property Name	Description
outboundTimestamp	Time when the client sends a network request
resourceURL	URL of the requested resource
responseCode	Response or status code for the network request
roundTripTime	Total duration of the transaction
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes
trackingID	Unique tracking identifier
userID	Unique user identifier
userAgent	Value of the User-Agent header
wifiAPS	Wireless access points
wifiConnectedMAC	MAC address of the connected wireless network
wifiConnectedSSID	Name of the connected wireless network
wifiTimestamp	Last recorded time that the client connected to the wireless network

Device App Push Action

Table 13-9. Device App Push Action Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
notificationAction	The action that was taken by the application that received the push notification. For example: open, received, seen
notificationID	Unique push notification identifier
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes
userID	Unique user identifier

Device App Session

Table 13-10. Device App Session Properties.

Property Name	Description
closedBy	Indicates what caused the end of the session. Valid values are USER and CRASH.
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
duration	The length of the usage session in milliseconds
mfpAppName	Name of the MobileFirst application
mfpAppVersion	Version of the MobileFirst application
timestamp	Time the session started, represented as epoch time stamp in milliseconds

Mobile Users

Table 13-11. Mobile Users Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
firstAccessTimestamp	Timestamp of the user's first access to a specific version of a MobileFirst application
timezone	The difference between UTC and local time, in minutes
userAuthBy	Name of the realm that the user authenticated with
userDisplayName	The display name of the user
userID	Unique user identifier

Server Logs

Table 13-12. Server Logs Properties.

Property Name	Description
appID	Unique Bluemix application identifier
hostname	Host name of the MobileFirst Analytics server

Table 13-12. Server Logs Properties (continued).

Property Name	Description
`level`	Level at which the log was recorded. For example: WARN, DEBUG
`loggerName`	Name of the logger
`message`	Message content of the log
`serverIpAddress`	IP address of the MobileFirst Analytics server
`sourceClass`	Name of the class that recorded the log
`sourceMethodName`	Name of the method that recorded the log
`stacktrace`	Stack trace that is associated with the log
`stacktraceMessage`	The message that was output by the exception that caused the stack trace
`timestamp`	Time the event was reported, represented as epoch time stamp in milliseconds
`timezone`	The difference between UTC and local time, in minutes
`trackingID`	Unique tracking identifier

Server Network Transactions

Table 13-13. Server Network Transactions Properties.

Property Name	Description
`adapterName`	Name of the adapter that was used
`appID`	Unique Bluemix application identifier
`bytesReceived`	Number of bytes that were received
`bytesSent`	Number of bytes that were sent
`clientID`	Unique client identifier
`compressed`	Boolean value that indicates whether the network payload was compressed
`deviceAppName`	Name of the MobileFirst application
`deviceAppVersion`	Version of the MobileFirst application
`deviceID`	Unique device identifier
`deviceModel`	Device model name. For example: iPhone3,3
`deviceOS`	Device environment. For example: iPhone
`deviceOSversion`	Device operating system version. For example: 7.0
`hostname`	Host name of the MobileFirst Analytics server
`inboundRequestURL`	URL of the inbound request
`inboundTimestamp`	Time when the reporting server received the client network request
`outboundTimestamp`	Time when the reporting server responds to the client network request
`procedureName`	Name of the adapter procedure

Table 13-13. Server Network Transactions Properties (continued).

Property Name	Description
serverIpAddress	IP address of the MobileFirst Analytics server
serverProcessingTime	Total amount of time the server took to respond
sessionID	Unique session identifier
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes
trackingID	Unique tracking identifier
userAgent	Value of the User-Agent header
userAuthBy	Name of the realm that the user authenticated with
userDisplayName	The display name of the user
userID	Unique user identifier

Server Push Notifications

Table 13-14. Server Push Notifications Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
destNumber	Phone number of the target recipient
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
eventSource	The push notification channel that generated the push notification
hostname	Host name of the MobileFirst Analytics server
mediator	Entity that processed the notification
messagePushTimestamp	Timestamp when the push notification was sent from the server
messageText	Message content of the push notification
notificationCountIOS	Number of push notifications that target recipients for iOS
notificationCountWindows	Number of push notifications that target recipients for Windows
notificationCountWeb	Number of push notifications that target recipients for Web

Table 13-14. Server Push Notifications Properties (continued).

Property Name	Description
notificationID	Unique push notification identifier
senderID	Identifier of the user who initiated the push notification
serverIpAddress	IP address of the MobileFirst Analytics server
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes

Server Push Subscriptions

Table 13-15. Server Push Subscriptions Properties.

Property Name	Description
appID	Unique Bluemix application identifier
clientID	Unique client identifier
deviceAppName	Name of the MobileFirst application
deviceAppVersion	Version of the MobileFirst application
deviceID	Unique device identifier
deviceModel	Device model name. For example: iPhone3,3
deviceOS	Device environment. For example: iPhone
deviceOSversion	Device operating system version. For example: 7.0
sessionID	Unique session identifier
timestamp	Time the event was reported, represented as epoch time stamp in milliseconds
timezone	The difference between UTC and local time, in minutes
userID	Unique user identifier

Navigating the Analytics Console

Learn how to use features that are provided by the MobileFirst Analytics Console.

Administration

Learn about the administrative aspects of the IBM MobileFirst Platform Operational Analytics.

Cluster Information:

The **Cluster Information** section of the **Cluster and Node** tab provides a list of properties about the state of the cluster.

Table 13-16. **Cluster Information** *properties*

Property Name	Description
Cluster Name	The unique name of the cluster that this node is a member of. Any node that joins the cluster must have the same cluster name set.
Cluster Status	Indicates the status of the cluster as GREEN, YELLOW, or RED. Green means that all shards and replicas are assigned and allocated. Yellow indicates that all primary shards are allocated, but some replicas are not. Red indicates that a shard is not allocated in the cluster.
Number of Nodes	Total number of nodes in the cluster.
Number of Master Nodes	Total number of master eligible nodes in the cluster.
Number of Data Nodes	Total number of non-master eligible nodes in the cluster (holds only data and processes queries, cannot be elected as a master node).
Number of Active Shards	Number of primary and replica shards that are assigned.
Number of Primary Shards	Number of primary shards (not including replicas).
Number of Unassigned Shards	Number of shards that are not yet allocated. This number is greater than zero if there are too many replica shards.

Update Settings:

The **Update Settings** page allows you to change settings dynamically. Typically these settings would need to be set via JNDI properties and require a server restart.

You can change these settings dynamically:
- Settings
- Documents Time To Live

The following table lists the **Settings** settings and their default values.

Table 13-17. **Settings** *settings for the Analytics Console Administration page.* This table lists the **Settings** settings and their default values for the Analytics Console Administration page.

Name	Default value
Ignore custom data	False
Ignore MFP application logs	False
Ignore MFP application push action	False
Ignore server logs	False
Ignore server network transactions	False
Ignore server push notifications	False
Ignore server push subscriptions	False

Table 13-17. **Settings** *settings for the Analytics Console Administration page (continued).* This table lists the **Settings** settings and their default values for the Analytics Console Administration page.

Name	Default value
Ignore devices	False
Ignore mobile users	False
Default tenant	worklight
Number of replicas	1

For more information about the **Settings** settings, see "Properties and configurations" on page 13-90.

The following table lists the **Documents Time To Live** settings.

Table 13-18. **Documents Time To Live** *settings for the Analytics Console Administration page.* This table lists the **Documents Time To Live** settings for the Analytics Console Administration page.

Event Type	Time To Live	Time To Live Unit
AnalyticsConfiguration	0	Minutes
CustomData	0	Minutes
MfpAppLogs	0	Minutes
ServerLogs	0	Minutes
ServerNetworkTransactions	0	Minutes
ServerPushNotifications	0	Minutes
ServerPushSubscriptions	0	Minutes

For more information about the **Documents Time To Live** settings, see "Data purging" on page 13-20.

Multi-tenancy:

Learn about multi-tenancy in the IBM MobileFirst Platform Operational Analytics.

Several MobileFirst Server instances 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 MobileFirst Server. If you want to use the same analytics cluster for multiple MobileFirst Server instances, 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 MobileFirst 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 name>/analytics-service/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 name>/analytics-service/data?tenant=test
```

To view the analytics data for a specific tenant, choose the tenant name from the drop-down list in the Analytics Console.

```
http://<host name>/analytics/console
```

Data purging:

Learn about purging data in the IBM MobileFirst Platform Operational Analytics.

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 13-6.

The following table shows the TTL properties:

Table 13-19. 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
`TTL_CustomData`	Time to live for custom data.
`TTL_MfpAppLogs`	Time to live for client logs, such as client-side captured logs, and stack traces.
`TTL_MfpAppPushAction`	Time to live for application push actions.
`TTL_ServerLogs`	Time to live for server logs.
`TTL_ServerNetworkTransactions`	Time to live for server network transactions.
`TTL_ServerPushNotifications`	Time to live for server push notifications.
`TTL_ServerPushSubscriptions`	Time to live for server push subscriptions.

Note: All JNDI properties must be preceded with the `analytics/` string. For more information about JNDI properties, see "JNDI properties" on page 13-90.

Three document types are not configured with a TTL property and therefore documents of this type are automatically purged:

- CustomCharts
- Devices
- MobileUsers

You can also set some of the TTL values in the Admin tab in the Analytics Console. Do not use both JNDI property and changing the setting in the console at the same time. A restart of the Analytics server reads and uses the JNDI property, which removes the previous value that was set in the console.

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 custom data TTL to one day in milliseconds:

```
<jndiEntry jndiName="analytics/TTL_CustomData" value="86400000" />
```

The following example shows how to set the client logs data TTL to five days:

```
<jndiEntry jndiName="analytics/TTL_MfpAppLogs" value="5d" />
```

The following example shows how to set the server logs TTL to one week:

```
<jndiEntry jndiName="analytics/TTL_ServerLogs" value="1w" />
```

Note: The TTL properties are not applied to data that 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 MobileFirst Platform Operational Analytics.

Exporting from the Administration page

The **Administration** page allows you to export all of the data stored in the analytics platform in one of the following formats:

- JSON
- CSV
- XML

This page is a user interface that displays the export feature and allows for the following options:

Table 13-20. Options for the export feature.

Option name	Description
Event Type	Name of the event that data will be exported from. For specific information about each of the events, see "Event types" on page 13-10.
Date Range	Start and end date for exporting data.
Limit	Number of analytics documents returned.
Offset	Number of documents to skip before exporting.
Data Format	Format data is exported in (JSON, CSV, or XML).

When an event type is selected from the user interface, you will have the option to select **Advanced Settings**. These settings allow you to apply additional filters to the data you are exporting.

Exporting custom data

The following example shows the format for exporting analytics data for custom data in JSON:

```
/administration/apps/{tenant}/export?query={"event":"CustomData","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion"
```

If the analytics console is hosted here:

```
http://hostname.ibm.com:9080
```

by using the curl command-line tool, the following curl command exports data from the `worklight` tenant for all versions of the `TestApp` application. Data is returned for only data for the iPhone environment, for all models of the iPhone, and only for iOS 7.0. The first 100 results that are found and start with the first result (limit = 100, offset = 0) are returned.

```
curl -XGET -G  'http://hostname.ibm.com::9080/analytics-service/data/administration/apps/worklight/export'
--data-urlencode   'query={"event":"ServerLogs","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","serverIpAddress":"*","level":"*"}' -u <username>
```

Replace the relevant values:

- **<username>:<password>**: your MobileFirst admin user name and password.
- **<output_file_name>**: the output file name.

Note: The default tenant is `worklight`. If you did not configure a specific tenant for the IBM MobileFirst Platform Operational Analytics, then you can use the `worklight` tenant. If you want to see data that occurred in a specific time range, specify a `startDate` time stamp in milliseconds and an `endDate` time stamp in milliseconds.

Exporting server push notifications

The following example shows the format for exporting analytics data for server push notifications:

```
/administration/apps/{tenant}/export?query={"event":"ServerPushNotifications","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","m
```

For example, by using the curl command-line tool, the following curl command exports data:

```
curl -XGET -G 'http://hostname.ibm.com:9080/analytics-service/data/administration/apps/worklight/export ' --data-urlencode
'query={"event":"ServerPushNotifications","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion":"*","mediator":"*","se
```

Exporting server logs

The following example shows the format for exporting analytics data for server logs:

```
/administration/apps/{tenant}/export?query={"event":"ServerLogs","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","serverIpAddress":"*","level":"*"}
```

For example, by using the curl command-line tool, the following curl command exports data:

```
curl -XGET -G 'http://hostname.ibm.com:9080/analytics-service/data/administration/apps/worklight/export ' --data-urlencode
'query={"event":"ServerLogs","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","serverIpAddress":"*","level":"*"}' -u <username>:<password> -o <output_file_name>
```

Note: Valid log levels for server logs include: FATAL, ERROR, WARN, INFO, LOG, DEBUG, and TRACE.

Exporting client logs

The following example shows the format for exporting analytics data for client logs:

```
/administration/apps/{tenant}/export?query={"event":"MfpAppLogs","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion":"*","deviceOS":"*","device
```

For example, by using the curl command-line tool, the following curl command exports data:

```
curl -XGET -G 'http://hostname.ibm.com:9080/analytics-service/data/administration/apps/worklight/export ' --data-urlencode
'query={"event":"MfpAppLogs","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion":"*","deviceOS":"*","deviceOSversion":"*","deviceModel":"*",
```

Note: Valid log levels for client logs include: ANALYTICS, FATAL, ERROR, WARN, INFO, LOG, DEBUG, and TRACE.

Exporting server network transactions

The following example shows the format for exporting analytics data for server network transactions:

```
/administration/apps/{tenant}/export?query={"event":"ServerNetworkTransactions","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion":"*","dev
```

For example, by using the curl command-line tool, the following curl command exports data:

```
curl -XGET -G 'http://hostname.ibm.com:9080/analytics-service/data/administration/apps/worklight/export ' --data-urlencode
'query={"event":"ServerNetworkTransactions","format":"json","limit":10,"offset":0,"startDate":"*","endDate":"*","mfpAppName":"*","mfpAppVersion":"*","deviceOS":"*","deviceOSversion":"*","de
```

Note: Valid log levels for client logs include: ANALYTICS, FATAL, ERROR, WARN, INFO, LOG, DEBUG, and TRACE.

Alerts:

If you are using the latest interim fix of MobileFirst, you can set thresholds in alert definitions in the IBM MobileFirst Analytics Console to better monitor your activities.

You can configure thresholds, which if exceeded, trigger alerts to notify the MobileFirst Analytics Console monitor. The triggered alerts can be visualized on the console, or the alerts can be handled by a custom webhook. This feature provides a proactive means of detecting client log errors, server log errors, extended periods of network latency, and authentication failures. Reactive thresholds and alerts keep you from having to sift through your data and set thresholds at a wide spectrum of granularity.

Creating an alert definition for client logs:

You can create an alert definition based on client logs.

Before you begin

Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive client logs.

About this task

In this example, you use client log data to create an alert definition. The alert monitors all client logs that were received in the last 5 minutes, and continues to check every 5 minutes, until the alert definition is disabled or deleted. An alert is triggered for each device that sent 3 or more client error logs with the same app name and version.

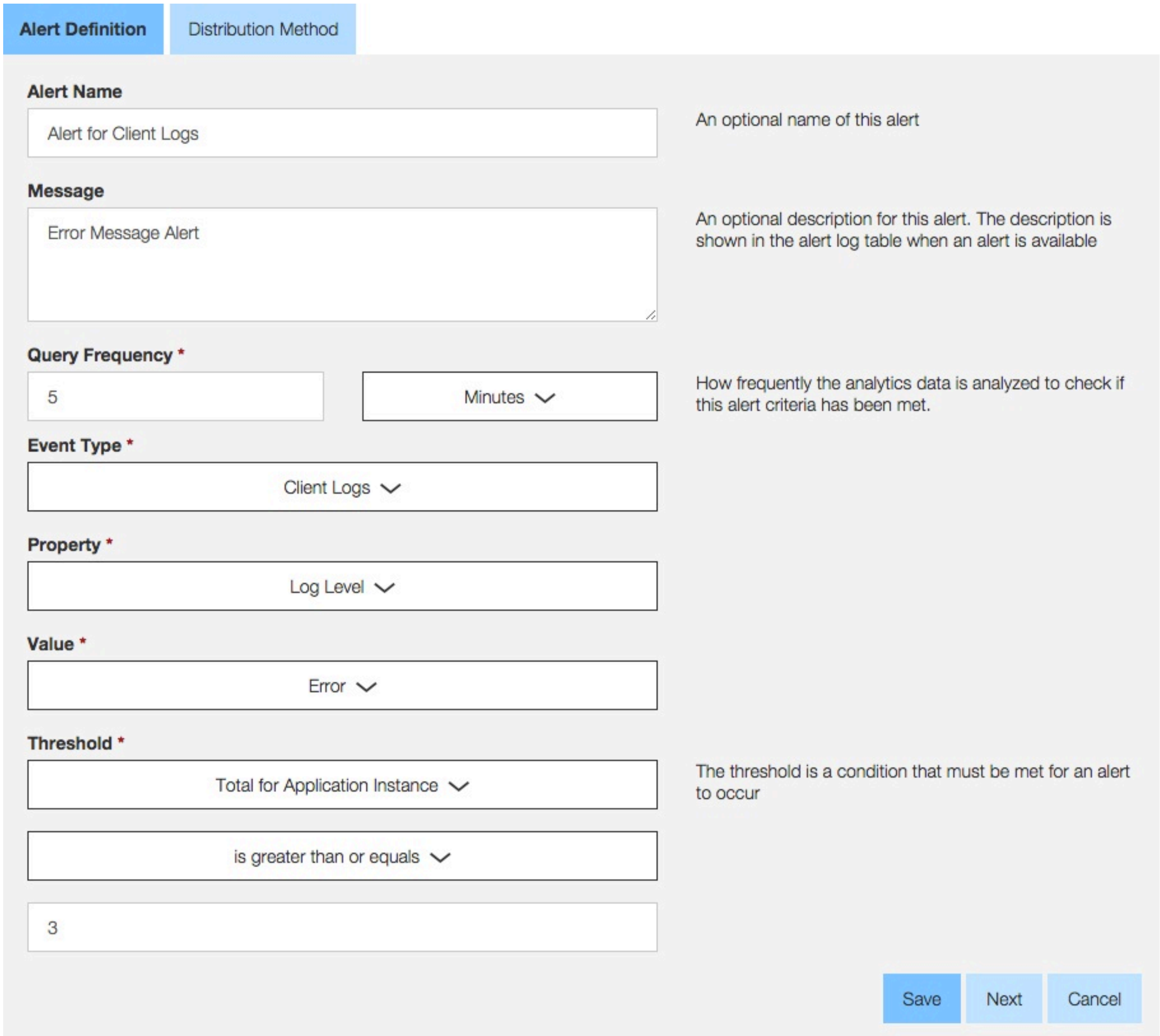
Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab and click **Create Alert**.

3. Provide the following values:
   - **Alert Name**: Alert for Client Logs
   - **Message**: Error Message Alert
   - **Query Frequency**: 5 Minutes
   - **Event Type**: Client Logs
     - **Property**: Log Level
       - **Value**: Error
       - **Threshold**
         - **Threshold Type**: Total for Application Instance

           Note: If you choose the **Average for Application** option, the client logs are averaged by the number of devices. For example, if you have two devices, and one device sends six client logs while the other device sends three client logs, the average is 4.5 client logs.
         - **Operator**: is greater than or equals 3

   The following image shows the alert definition tab:



Alert Name

Alert for Client Logs

An optional name of this alert

Message

Error Message Alert

An optional description for this alert. The description is shown in the alert log table when an alert is available

Query Frequency *

5

Minutes

How frequently the analytics data is analyzed to check if this alert criteria has been met.

Event Type *

Client Logs

Property *

Log Level

Value *

Error

Threshold *

Total for Application Instance

is greater than or equals

3

The threshold is a condition that must be met for an alert to occur

4. Click the **Distribution Method** tab, and provide the following value:
   - **Method**: Analytics Console Only

     Note: Choose the **Analytics Console and Network Post** option if you want to additionally send a POST message with a JSON payload to your customized URL. The following fields are available if you choose this option:
     - **Network Post Url** (required)
     - **Headers** (optional)
     - **Authentication Type** (required)
5. Click **Save**.

Results

You created an alert definition to trigger an alert at the end of each 5-minute interval if the number of client logs reached your threshold of 3 or more error logs.

Creating an alert definition for server logs:

You can set an alert based on server logs.

Before you begin

Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive logs.

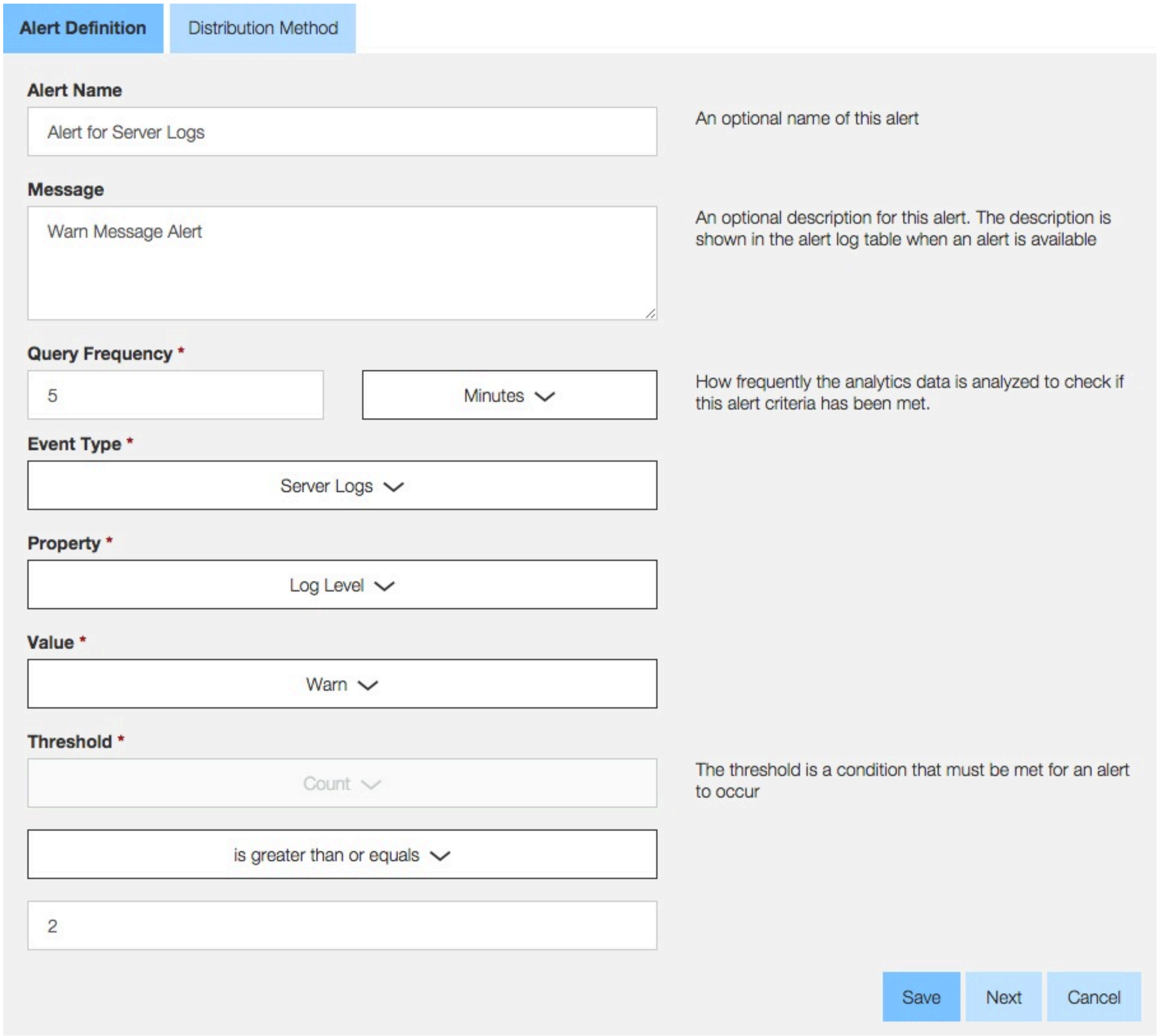
About this task

In this example, you use server log data to create an alert definition. The alert monitors all server logs that were received in the last 5 minutes, and continues to check every 5 minutes, until the alert definition is disabled or deleted. An alert is triggered if there are 2 or more server warn logs.

Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab and click **Create Alert**.
3. Provide the following values:
   - **Alert Name**: Alert for Server Logs
   - **Message**: Warn Message Alert
   - **Query Frequency**: 5 Minutes
   - **Event Type**: Server Logs
     - **Property**: Log Level
       - **Value**: Warn
       - **Threshold**
         - **Threshold Type**: Count
         - **Operator**: is greater than or equals 2

   The following image shows the alert definition tab:



Alert Name

Alert for Server Logs

An optional name of this alert

Message

Warn Message Alert

An optional description for this alert. The description is shown in the alert log table when an alert is available

Query Frequency *

5

Minutes

How frequently the analytics data is analyzed to check if this alert criteria has been met.

Event Type *

Server Logs

Property *

Log Level

Value *

Warn

Threshold *

Count

is greater than or equals

2

The threshold is a condition that must be met for an alert to occur

4. Click the **Distribution Method** tab, and provide the following value:
   - **Method**: Analytics Console Only

     Note: Choose the **Analytics Console and Network Post** option if you want to additionally send a POST message with a JSON payload to your customized URL. The following fields are available if you choose this option:
     - **Network Post Url** (required)
     - **Headers** (optional)
     - **Authentication Type** (required)
5. Click **Save**.

Results

You created an alert definition to trigger an alert at the end of each 5-minute interval if the number of server logs reached your threshold of 2 or more warn logs.

Creating an alert definition for network transactions:

You can create an alert definition based on server network transactions.

Before you begin

Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive analytics data.

About this task

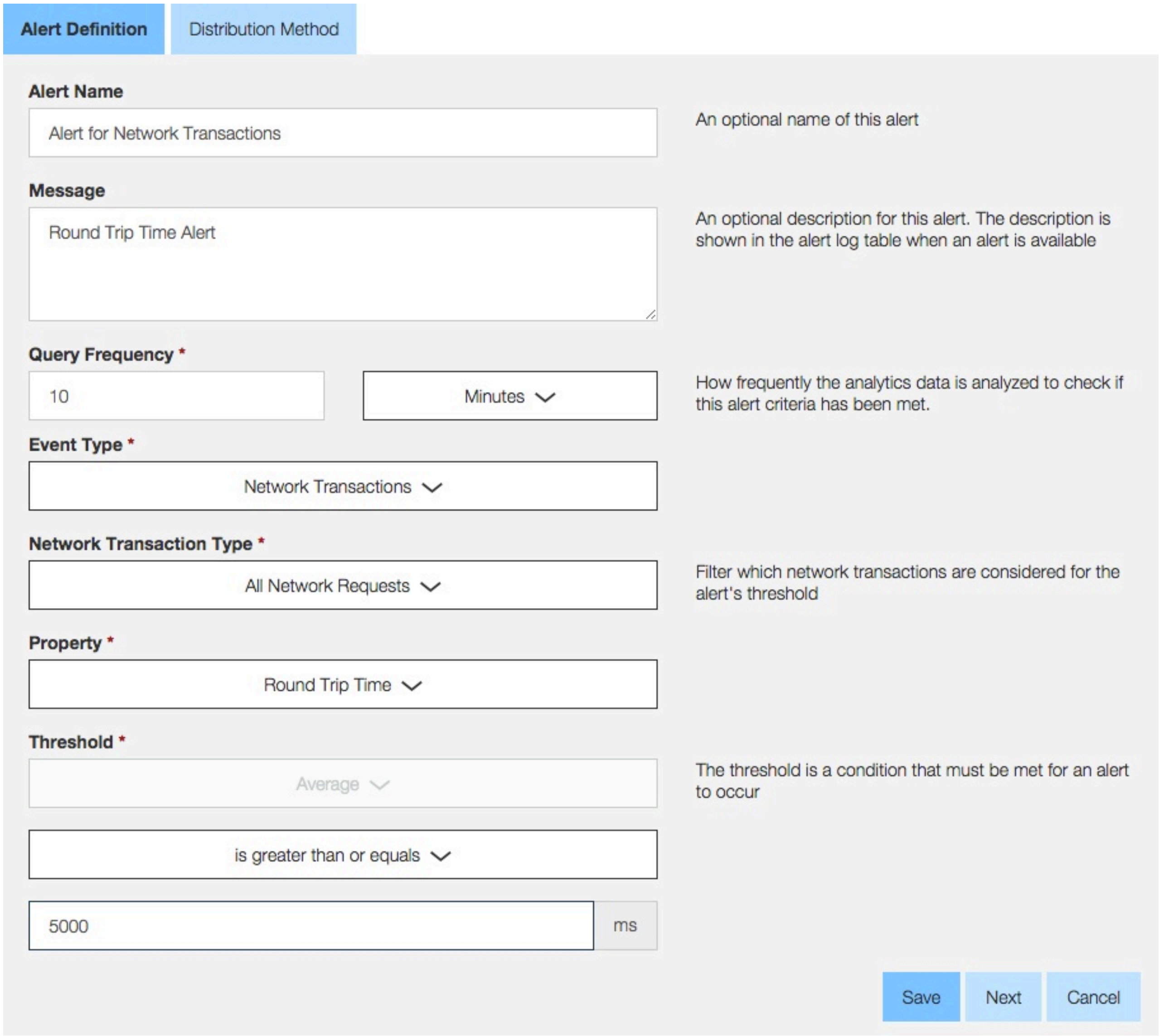
In this example, you use server network transaction data to create an alert definition. The alert monitors all network transactions in the last 10 minutes, and continues to check every 10 minutes. An alert is triggered when the average round-trip time exceeds 5 seconds, until the alert is disabled or deleted.

Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab and click **Create Alert**.
3. Provide the following values:
   - **Alert Name**: Alert for Network Transactions
   - **Message**: Round Trip Time Alert
   - **Query Frequency**: 10 Minutes
   - **Event Type**: Network Transactions
     - **Network Transaction Type**: All Network Requests

       Note: You can also choose **All Adapter Requests** or **Specific Adapter Requests**. If you choose **Specific Adapter Requests**, you must also provide a value for **Adapter**.
     - **Property**: Round Trip Time
       - **Threshold**
         - **Threshold Type**: Average
         - **Operator**: is greater than or equals 5000 ms

   The following image shows the alert definition tab:



Alert Name

Alert for Network Transactions

An optional name of this alert

Message

Round Trip Time Alert

An optional description for this alert. The description is shown in the alert log table when an alert is available

Query Frequency *

10 | Minutes

How frequently the analytics data is analyzed to check if this alert criteria has been met.

Event Type *

Network Transactions

Network Transaction Type *

All Network Requests

Filter which network transactions are considered for the alert's threshold

Property *

Round Trip Time

Threshold *

Average

is greater than or equals

5000 ms

The threshold is a condition that must be met for an alert to occur

4. Click the **Distribution Method** tab, and provide the following value:
   - **Method**: Analytics Console Only

     Note: Choose the **Analytics Console and Network Post** option if you want to additionally send a POST message with a JSON payload to your customized URL. The following fields are available if you choose this option:
     - **Network Post Url** (required)
     - **Headers** (optional)
     - **Authentication Type** (required)
5. Click **Save**.

Results

You created an alert definition to trigger an alert at the end of each 10-minute interval when the average round-trip time for all network requests reached your threshold of 5 or more seconds.

Creating an alert definition for authorization failures:

You can create an alert definition based on authorization failures.

Before you begin

Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive logs.

About this task

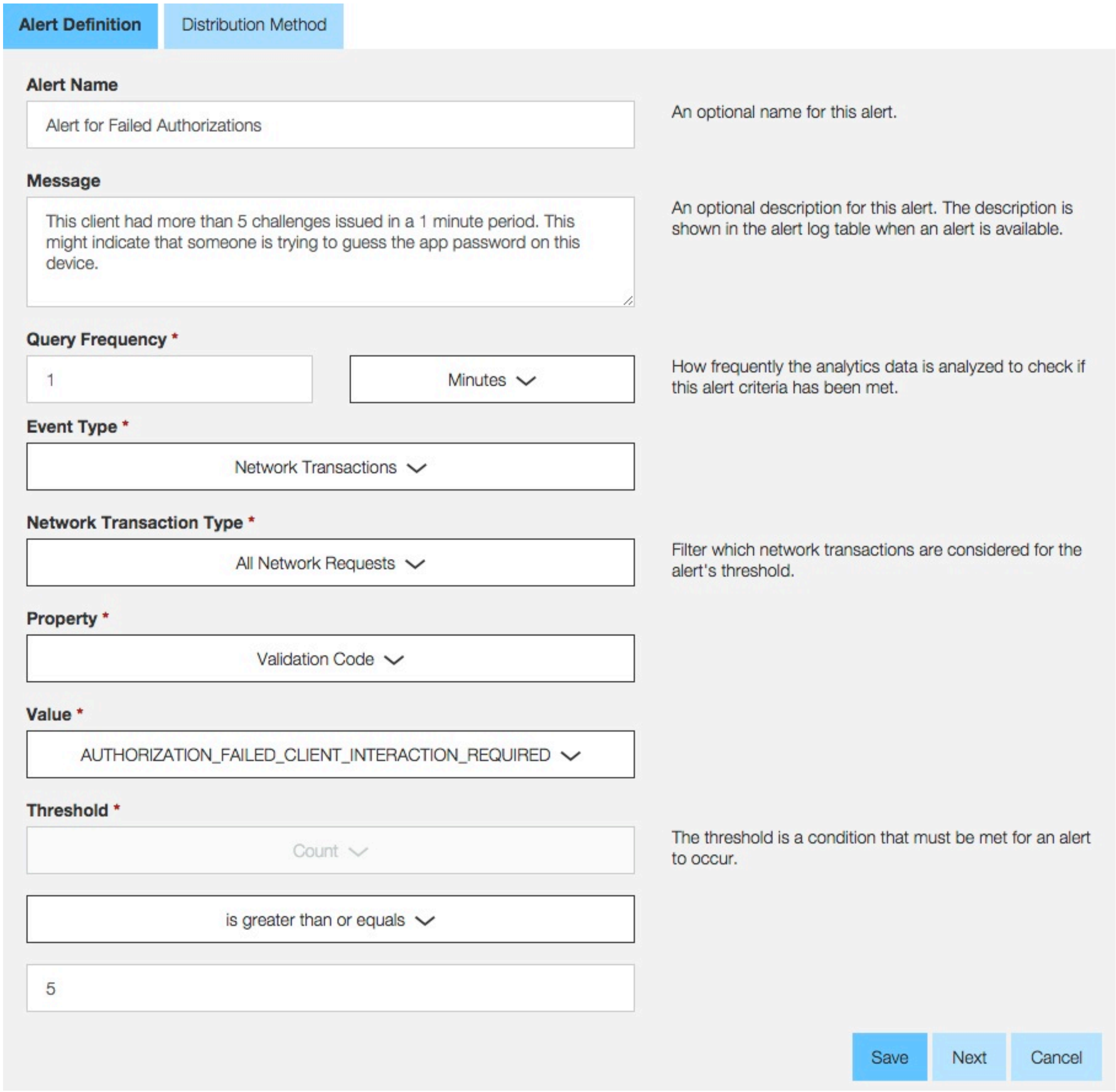
In this example, you use validation code data to create an alert definition. The alert monitors all network transactions in the last minute, and continues to check every minute, until the alert definition is disabled or deleted. An alert is triggered when the number of failed authorizations exceeds 5.

Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab and click **Create Alert**.
3. Provide the following values:
   - **Alert Name**: Alert for Failed Authorization
   - **Message**: This client had more than 5 challenges issued in a 1 minute period. This might indicate that someone is trying to guess the app password on this device.
   - **Query Frequency**: 1 Minutes
   - **Event Type**: Network Transactions
     - **Network Transaction Type**: All Network Requests

       Note: You can also choose **All Adapter Requests** or **Specific Adapter Requests**. If you choose **Specific Adapter Requests**, you must also provide a value for **Adapter**.
     - **Property**: Validation Code
       - **Value**: AUTHORIZATION_FAILED_CLIENT_INTERACTION_REQUIRED
       - **Threshold**
         - **Threshold Type**: Count
         - **Operator**: is greater than or equals 5

   The following image shows the alert definition tab:



Alert Name

Alert for Failed Authorizations

An optional name for this alert.

Message

This client had more than 5 challenges issued in a 1 minute period. This might indicate that someone is trying to guess the app password on this device.

An optional description for this alert. The description is shown in the alert log table when an alert is available.

Query Frequency *

1 Minutes

How frequently the analytics data is analyzed to check if this alert criteria has been met.

Event Type *

Network Transactions

Network Transaction Type *

All Network Requests

Filter which network transactions are considered for the alert's threshold.

Property *

Validation Code

Value *

AUTHORIZATION_FAILED_CLIENT_INTERACTION_REQUIRED

Threshold *

Count

is greater than or equals

5

The threshold is a condition that must be met for an alert to occur.

4. Click the **Distribution Method** tab and provide the following values:
   - **Method**: Analytics Console and Network Post

     Note: Choose the **Analytics Console Only** option if you do not want to additionally send a POST message with a JSON payload to your customized URL.
   - **Network Post Url** `http://myHost.com:5000/myEmailEndPoint`

     Note: You must provide a valid endpoint URL to receive a POST message.
   - **Authentication Type** Anonymous
5. Click **Save**.

Results

You created an alert definition to trigger an alert and send a POST message to your endpoint URL at the end of each 1-minute interval when the number of failed authorizations reached your threshold of 5 or more failures.

Example

The following example shows the POST message that is sent to your network post URL:

```
2015-09-21 10:15:04 - POST request at /myEmailEndPoint with
body {"message":"This client had more than 5 challenges issued in a 1 minute period. This might indic
                 is trying to guess the app password on this device.",
      "timestamp":1442848504431,
      "title":"Alert for Failed Authorization",
      "condition":{"value":5.0,"operator":"GTE"},
      "value":"AUTHORIZATION_FAILED_CLIENT_INTERACTION_REQUIRED",
      "offenders":{"ChallengeApp 1.0 6c1fc633-5c78-88bb-c5a3-3de257bdbade":5.0},
      "property":"validationCode",
      "eventType":"ServerNetworkTransactions"}
form data {}
and headers:
User-Agent - Java/1.7.0_71
Content-Length - 473
Pragma - no-cache
Host - myHost.com:5000
Cache-Control - no-cache
Accept - text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Content-Type - application/json
Connection - keep-alive
```

Creating an alert definition for application crashes:

You can create an alert definition based on application crashes.

Before you begin

Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive client logs.

About this task

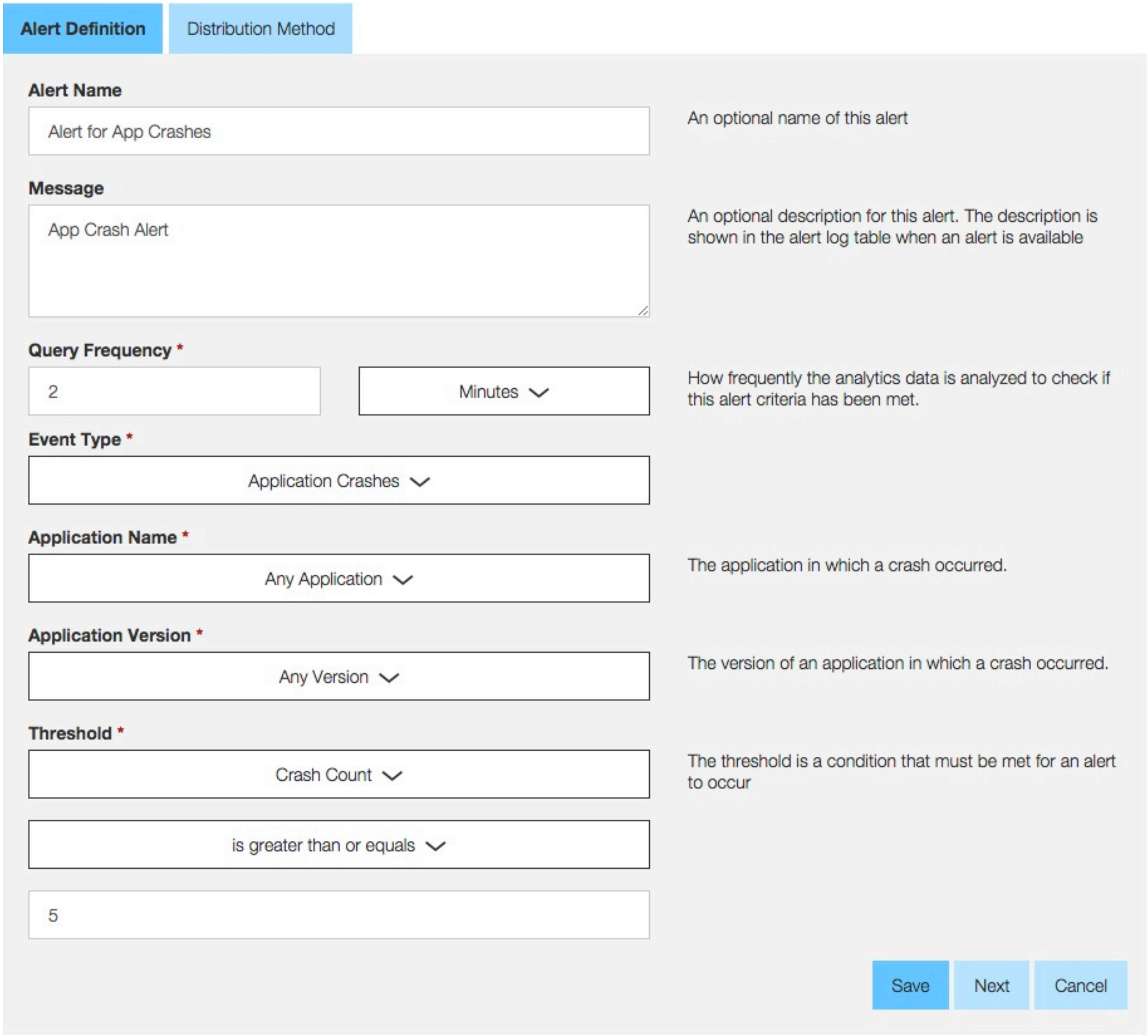
In this example, you use app crash data to create an alert definition. The alert monitors all app crashes in the last 2 minutes, and continues to check every 2 minutes, until the alert definition is disabled or deleted. An alert is triggered for each application that crashed 5 or more times.

Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab and click **Create Alert**.
3. Provide the following values:
   - **Alert Name**: Alert for App Crashes
   - **Message**: App Crash Alert
   - **Query Frequency**: 2 Minutes
   - **Event Type**: Application Crashes
     - **Application Name**: Any Application

  - **Application Version**: Any Version
  - **Threshold**
    - **Threshold Type**: Crash Count
    - **Operator**: is greater than or equals 5

The following image shows the alert definition tab:



Alert Name

Alert for App Crashes

An optional name of this alert

Message

App Crash Alert

An optional description for this alert. The description is shown in the alert log table when an alert is available

Query Frequency *

2

Minutes

How frequently the analytics data is analyzed to check if this alert criteria has been met.

Event Type *

Application Crashes

Application Name *

Any Application

The application in which a crash occurred.

Application Version *

Any Version

The version of an application in which a crash occurred.

Threshold *

Crash Count

is greater than or equals

5

The threshold is a condition that must be met for an alert to occur

4. Click the **Distribution Method** tab, and provide the following value:
   - **Method**: Analytics Console Only

     Note: Choose the **Analytics Console and Network Post** option if you want to additionally send a POST message with a JSON payload to your customized URL. The following fields are available if you choose this option:
     - **Network Post Url** (required)
     - **Headers** (optional)
     - **Authentication Type** (required)
5. Click **Save**.

Results

You created an alert definition to trigger an alert at the end of each 2-minute interval if the number of app crashes reached your threshold of 5 or more crashes.

Managing alert definitions:

You can manage your alert definitions.

Before you begin

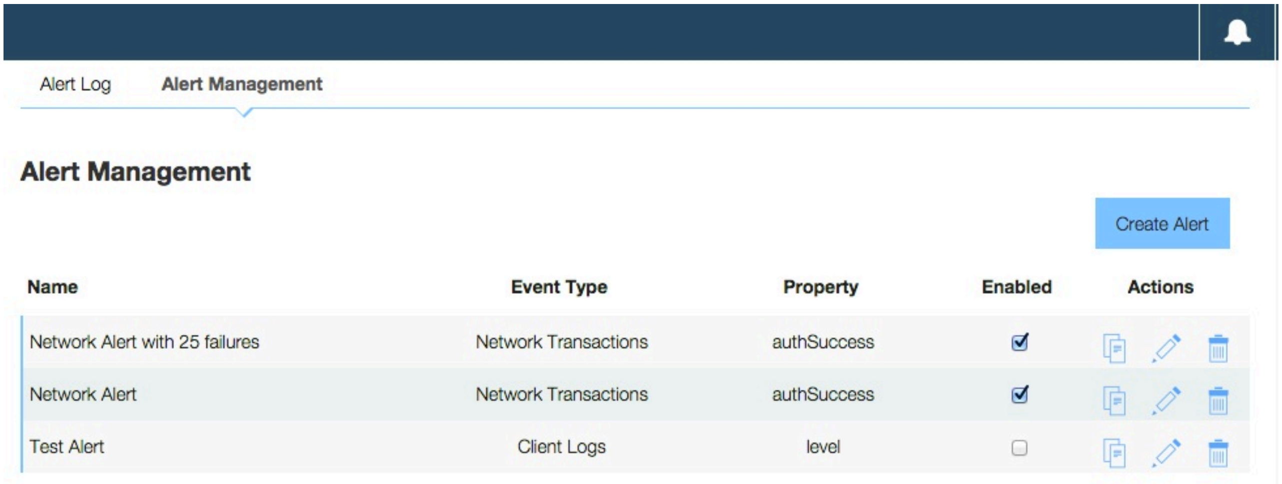
Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive analytics data.

About this task

In this example, you manage your alert definitions from the Alert Management page.

Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page.
2. Click the **Alert Management** tab. The following image shows the Alert Management page:



Alert Management

Name	Event Type	Property	Enabled	Actions
Network Alert with 25 failures	Network Transactions	authSuccess	☑	
Network Alert	Network Transactions	authSuccess	☑	
Test Alert	Client Logs	level	□	

3. Optional: Toggle the check-box under the **Enabled** column to enable or disable a specific alert definition.
4. Optional: Click the **Duplicate** icon if you want to create a copy of an alert definition and change some values.
5. Optional: Click the **Pencil** icon if you want to edit an alert definition.
6. Optional: Click the **Trash** icon if you want to delete an alert definition.

Results

You managed your alert definitions.

Viewing alert details:

You can view the details of your triggered alerts.

Before you begin

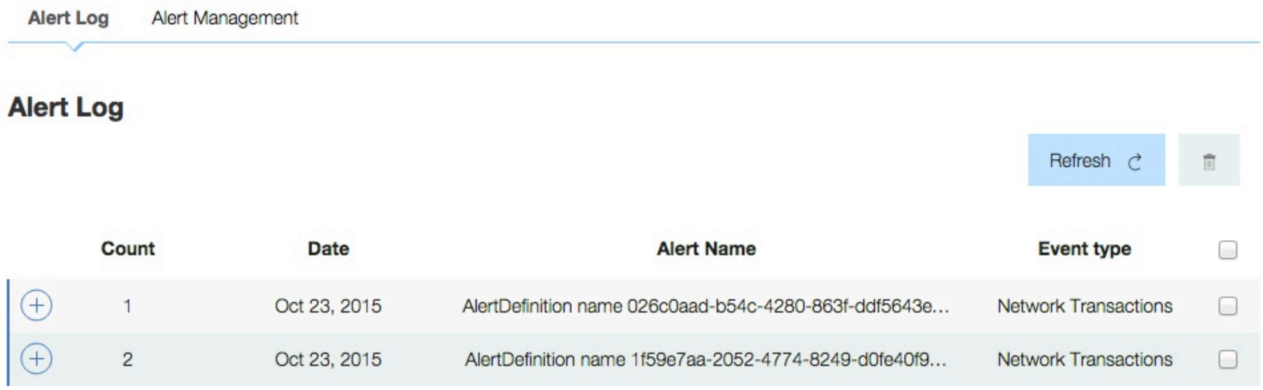
Ensure that the IBM MobileFirst Platform Operational Analytics Server is started and ready to receive analytics data.

About this task

In this example, you view the details of your triggered alerts from the Alert Log page.

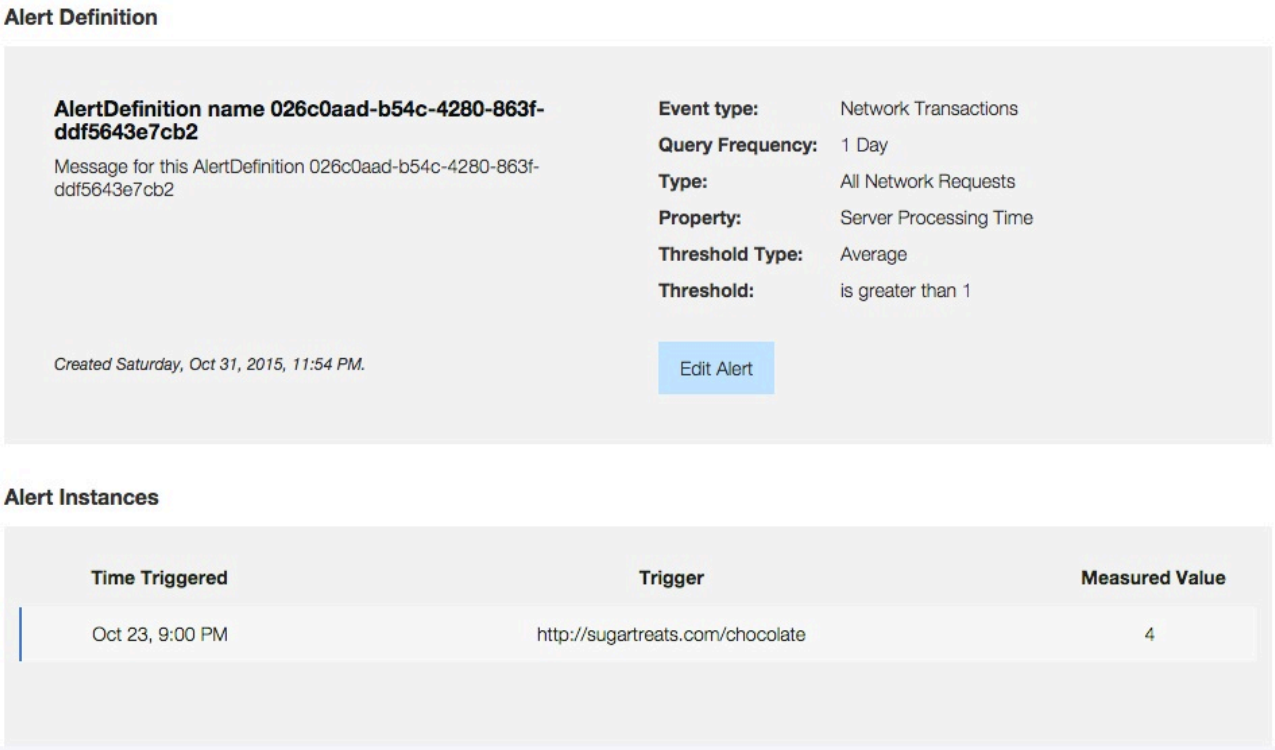
Procedure

1. In the MobileFirst Analytics Console, click the **Alerts** icon. This action brings up the Alert Log page. The following image shows the Alert Log page:



	Count	Date	Alert Name	Event type	□
⊕	1	Oct 23, 2015	AlertDefinition name 026c0aad-b54c-4280-863f-ddf5643e...	Network Transactions	□
⊕	2	Oct 23, 2015	AlertDefinition name 1f59e7aa-2052-4774-8249-d0fe40f9...	Network Transactions	□

2. Click the **+** icon for any of the alerts. This action displays the **Alert Definition** and **Alert Instances** sections. The following image shows the **Alert Definition** and **Alert Instances** sections:



Alert Definition

AlertDefinition name 026c0aad-b54c-4280-863f-ddf5643e7cb2

Message for this AlertDefinition 026c0aad-b54c-4280-863f-ddf5643e7cb2

Created Saturday, Oct 31, 2015, 11:54 PM.

Event type:	Network Transactions
Query Frequency:	1 Day
Type:	All Network Requests
Property:	Server Processing Time
Threshold Type:	Average
Threshold:	is greater than 1

Alert Instances

Time Triggered	Trigger	Measured Value
Oct 23, 9:00 PM	http://sugartreats.com/chocolate	4

Note: If the corresponding alert definition was not deleted or modified, you can edit the alert definition by clicking **Edit Alert**. Otherwise, the **Edit Alert** button is unavailable and the following message is displayed:

```
This alert definition has since been modified or deleted.
```

3. Optional: Select an alert and click the **Trash** icon to delete the alert.

Results

You viewed more details about your alerts.

Application crashes:

If you are using the latest interim fix of MobileFirst, you can view information about your application crashes in the IBM MobileFirst Analytics Console to better monitor and troubleshoot your applications.

Application crash monitoring:

You can quickly see information about your app crashes in the Dashboard page of the IBM MobileFirst Analytics Console.

Two new charts are available: **Crash Overview** and **Crashes**.

The **Crash Overview** table shows the following data columns:

- **Application**: application name
- **Crashes**: total number of crashes for that application
- **Total Uses**: total number of times a user opens and closes that application
- **Crash Rate**: percentage of crashes per use

The **Crashes** bar graph shows a histogram of crashes over time. The data can be shown in two ways:

- **Display crash rate**: crash rate over time
- **Display total crashes**: total crashes over time

Note: Both charts query against the `MfpAppSession` documents. You must instrument your application to collect app uses and crashes for data to appear in the charts. If `MfpAppSession` data is not collected, then `MfpAppLog` documents are queried. In this case, the charts can count the number of crashes, but cannot compute a crash rate because the number of app uses is unknown, which results in the following limitations:

- The **Crash Overview** table has empty columns for **Total Uses** and **Crash Rate**.
- The **Crashes** bar graph displays no data when **Display Crash Rate** is selected.

To instrument crash data, see "Instrumenting your iOS application for application crash data."

Application crash troubleshooting:

You can view the **Crash Summary** tab in the Dashboard page of the IBM MobileFirst Analytics Console to better administer your applications.

The **Crash Summary** tab displays a sortable table with the following data columns:

- **Crashes**
- **Devices**
- **Last Crash**
- **Application**
- **OS**
- **Message**

You can click on the **+** icon next to any entry to display the **Crash Details** table, which includes the following columns:

- **Time Crashed**
- **Application Version**
- **OS Version**
- **Device Model**
- **Device ID**
- **Download**: link to download the logs that led up to the crash

You can expand any entry in the **Crash Details** table to get more details, including a stacktrace.

Note: The data for the **Crash Summary** tab is populated by querying the fatal level client logs. If your application does not collect fatal client logs, no data is available.

Instrumenting your iOS application for application crash data:

You must instrument your iOS application code to collect application crash data.

Before you begin

To capture sessions that are closed by a crash and by the user, you must use the iOS Logger API in the application. You must also send both analytics and logs with the `OCLogger` and `WLAnalytics` APIs.

About this task

In this example, you instrument your iOS application to collect app crash data.

Procedure

Add the following call to the beginning of your application code in `application:didFinishLaunchingWithOptions:` in the app delegate.

```
[[WLAnalytics sharedInstance] startRecordingApplicationLifecycleEvents]
```

Results

You instrumented your iOS application for app crash data.

Custom Analytics

Starting with V7.0.0, you can visualize the collected analytics data in your analytics repository. This visualization is a powerful way to inspect data for specific use cases. You can create charts with data already collected by Operational Analytics in addition to custom data you report.

The ability to report custom data (custom key-value pairs) from the IBM MobileFirst Platform Foundation for iOS was present since V6.1.0. Now you can extract that data in raw form, and render custom charts from that raw data.

Creating a custom chart:

MobileFirst Operational Analytics has several pre-made charts with useful visualizations. Starting with V7.1.0, you can create a different chart type or run a specific query to render into a table.

About this task

In this scenario, you create a bar chart for all active Android Galaxy devices.

Procedure

1. Click the **Custom Charts** tab from the Dashboard page of the MobileFirst Analytics Console.

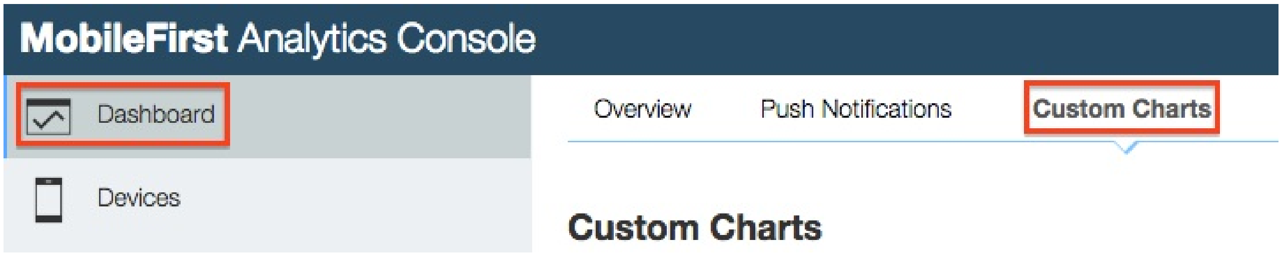


Figure 13-1. Analytics Dashboard

2. Click **Create Chart**. This action opens a **General Settings** tab where you specify your chart title and event type.

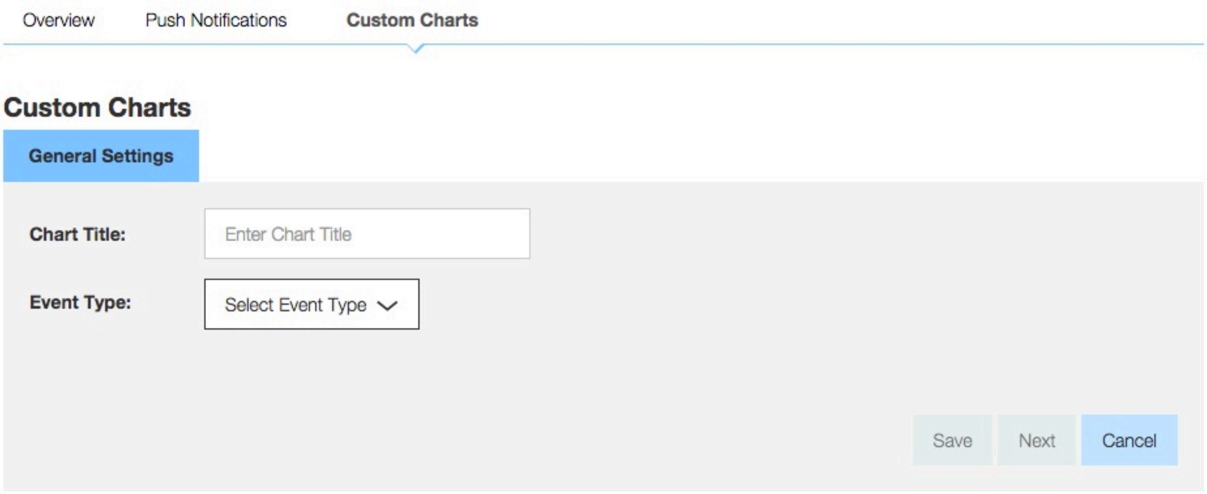


Figure 13-2. General Settings

   a. Enter a chart title.
   b. Choose the event type. For this scenario, choose **Server Network Transactions**.
   c. Choose a chart type. For this scenario, choose **Bar Graph**.

3. Click the **Chart Definition** tab.

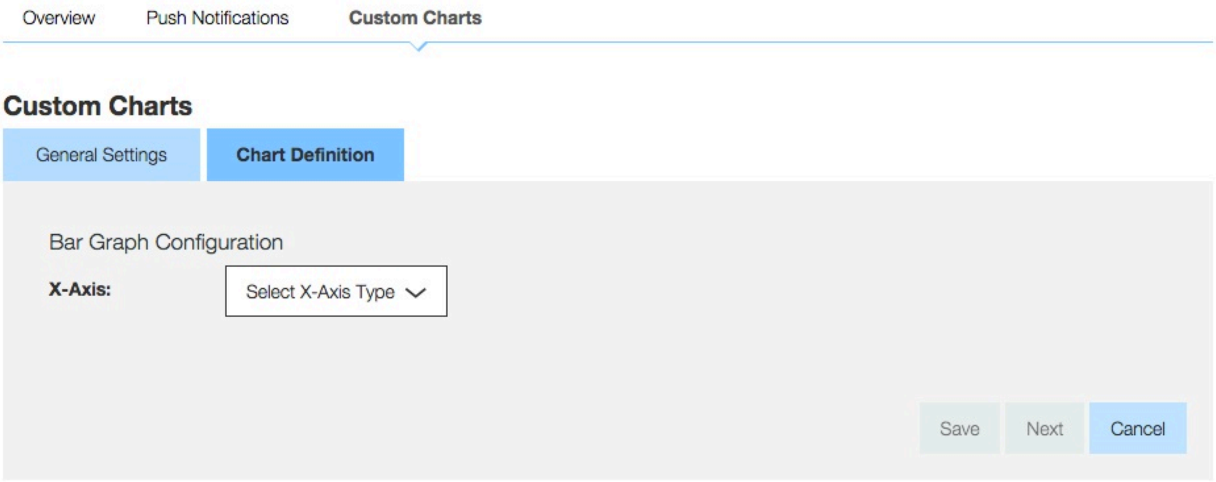


Figure 13-3. Chart Definition

   a. Select **Axis by Timeline** for the x-axis.
   b. Select **Unique** for the y-axis.
   c. Select **Device Id** for the property.

4. Click the **Chart Filters** tab to specify filters for the data.

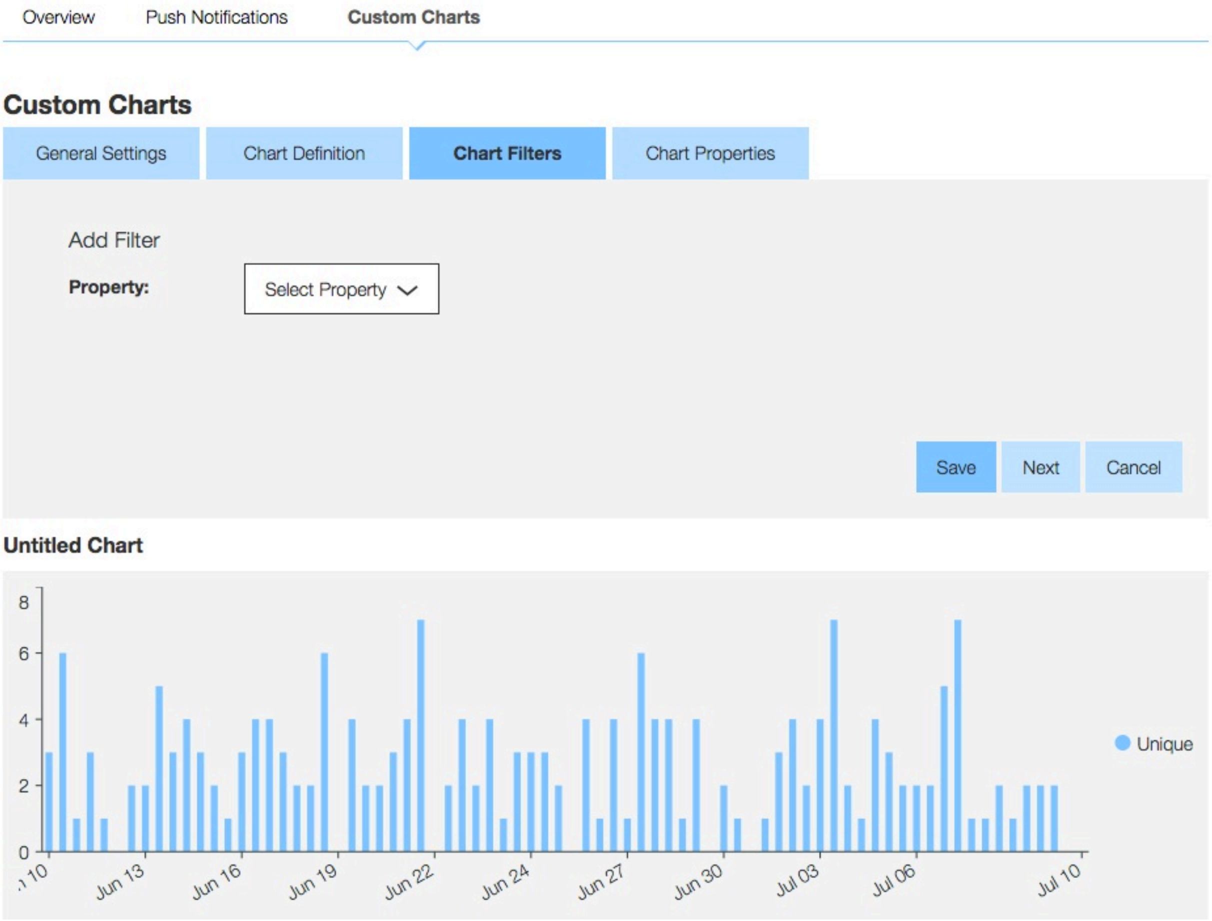


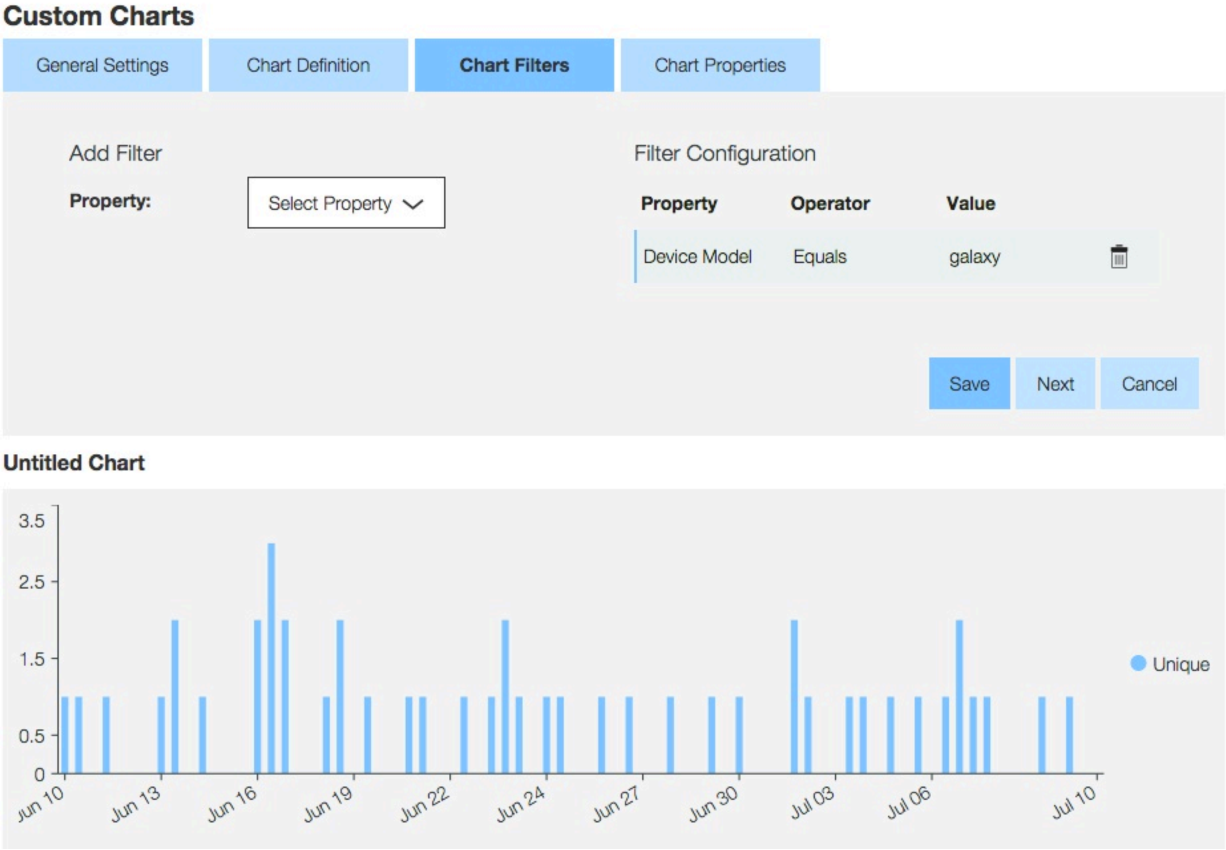
Figure 13-4. Chart Filters

a. Select the property to filter. For this scenario, choose **Device Model**.

b. Select the operator to filter. For this scenario, choose **Equals**.

c. Select the value to filter on. For this scenario, choose **galaxy**.

d. Click **Add Filter**.



Property: Select Property

Property	Operator	Value
Device Model	Equals	galaxy

Figure 13-5. Chart Filters

5. Click the **Chart Properties** tab.

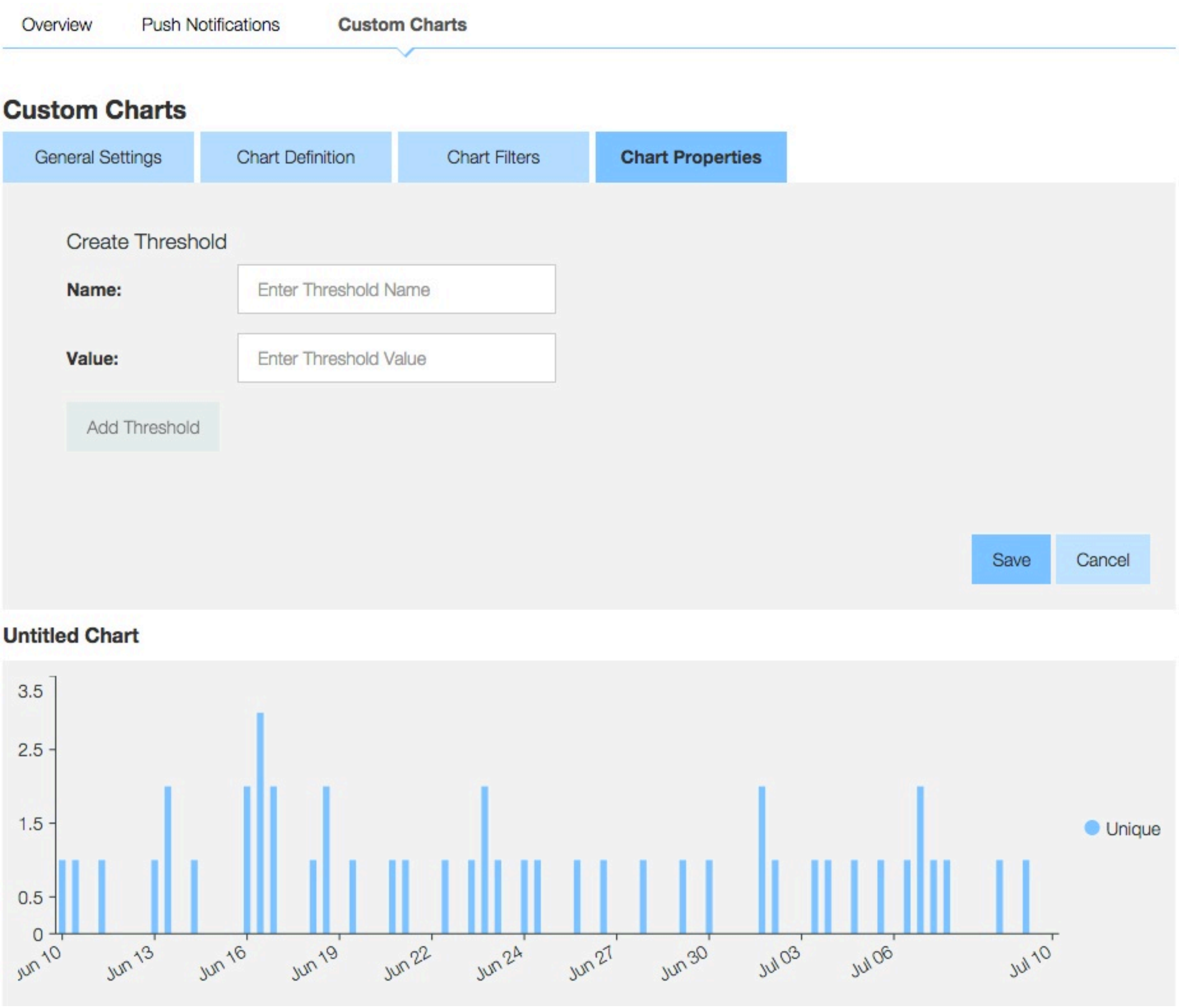


Figure 13-6. Chart Properties

a. Enter a threshold name.
b. Enter a threshold value.

6. Click **Save**. Repeat steps 1-5 for all filters that you want to apply.

What to do next

For more advanced usage, see "Patterns for visualizing custom data" on page 13-55.

Custom chart types:

The MobileFirst Analytics Console offers the following chart types for data visualization.

- Bar Graph
- Flow Chart
- Line Graph
- Metric Group
- Pie Chart

- Table

Note: The following sections contain only information about custom chart types that might not be familiar.

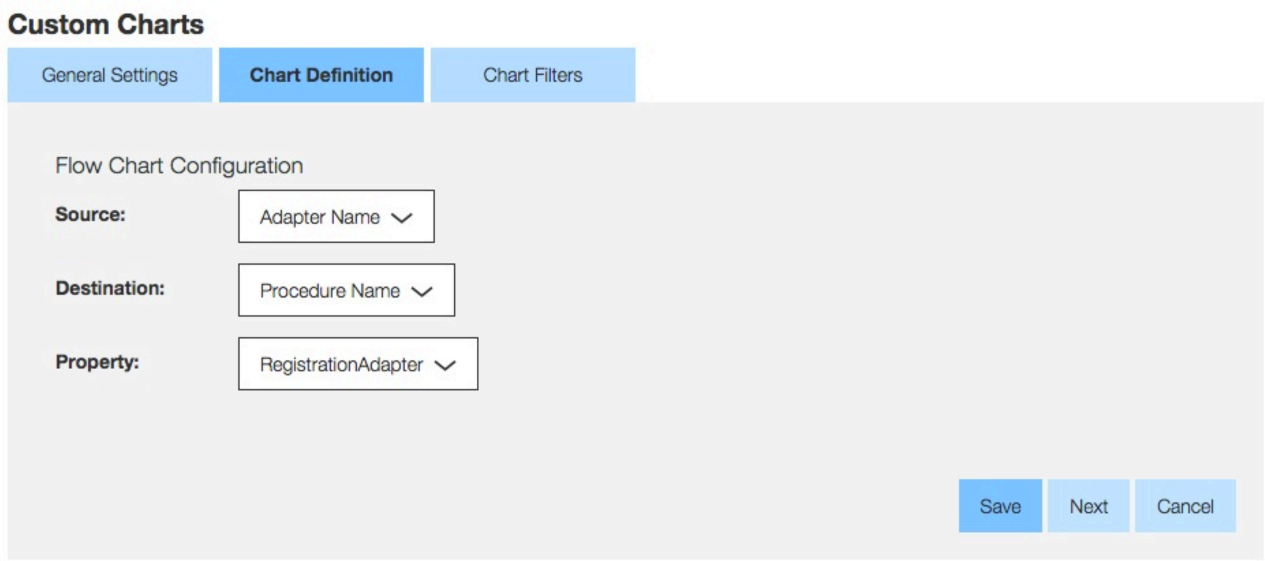
Flow Chart:

The flow chart can be used to show relationships between two different properties.

Three selections must be made to use the flow chart:

- A source
- A destination
- A value from either the source or destination

If you want to see the relationship between an adapter and its procedures, you select **Adapter Name** as the source and **Procedure Name** as the destination. You then select one of the adapters as the value.



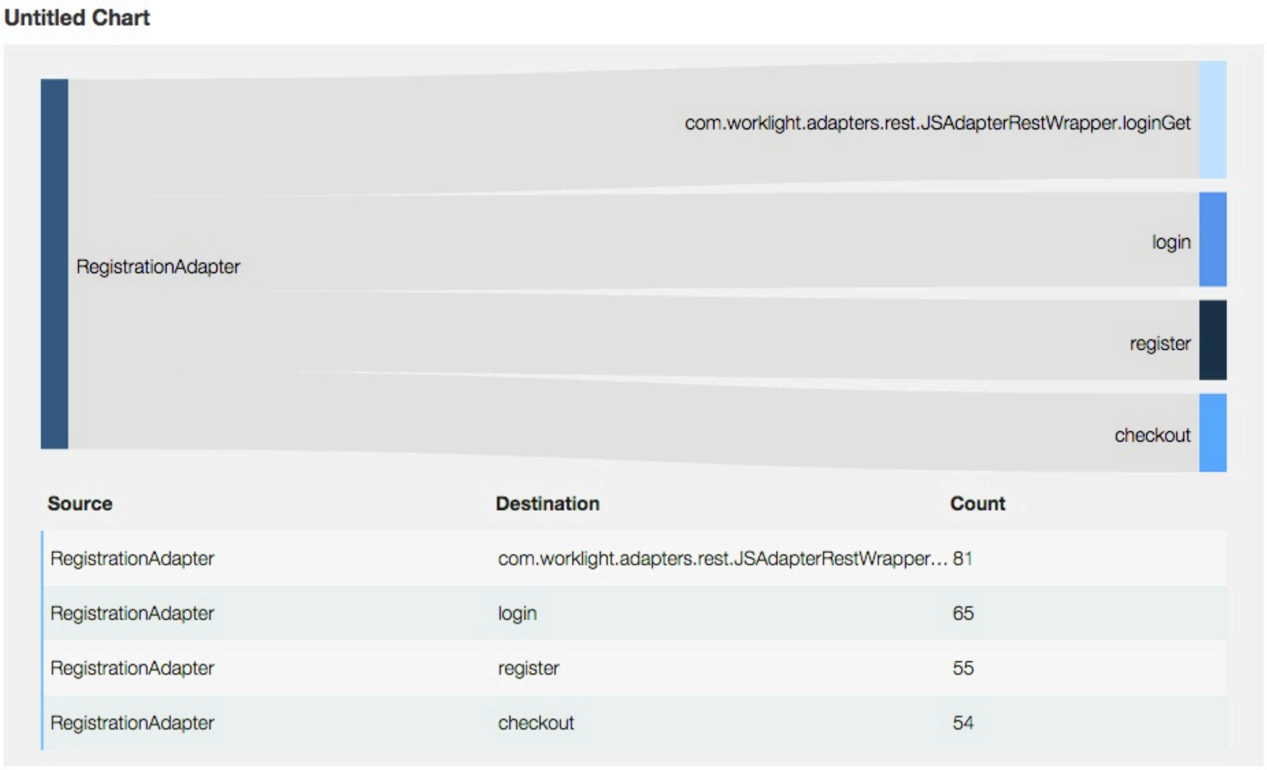
Flow Chart Configuration

Source: Adapter Name

Destination: Procedure Name

Property: RegistrationAdapter

Figure 13-7. Adapter-Procedure chart definition



Source	Destination	Count
RegistrationAdapter	com.worklight.adapters.rest.JSAdapterRestWrapper…	81
RegistrationAdapter	login	65
RegistrationAdapter	register	55
RegistrationAdapter	checkout	54

Figure 13-8. Adapter-Procedure Flow Chart

The RegistrationAdapter adapter is the source on the left and its four procedures are on the right. The line next to each procedure represents a relationship between the source and each of its destinations. The length of this line is relative to the frequency at which that procedure was called. In this case, all four of the RegistrationAdapter procedures were called about the same number of times. A table is displayed following the flow chart with the exact values of each relationship.

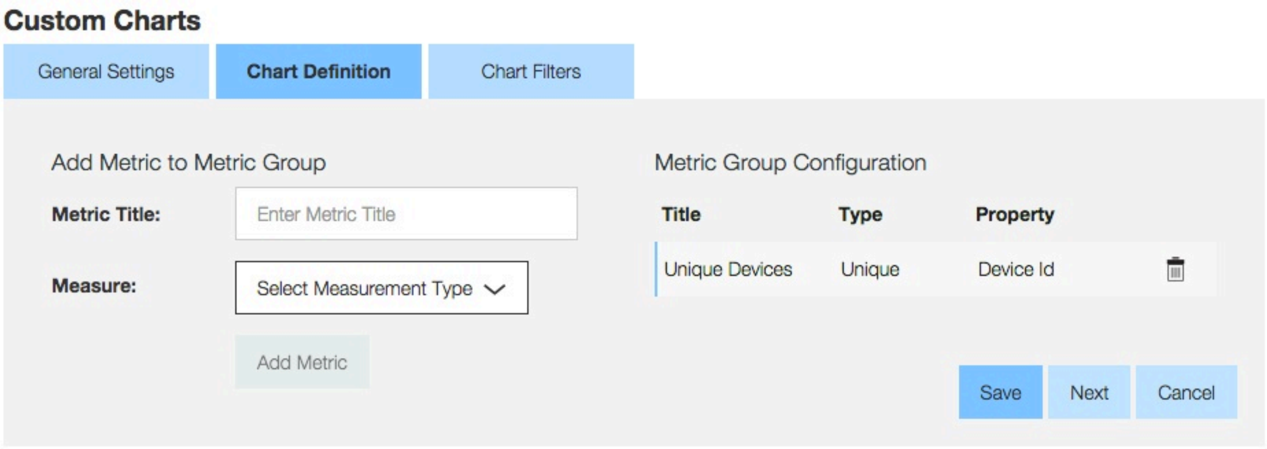
Metric Group:

The Metric Group chart is used to display a single numerical value.

This value can be:

- A total or sum of events
- An average of a specified property
- A unique count

If you want to see an at-a-glance view of how many unique devices contacted your server, you can make the following selections.



Metric Title: Enter Metric Title

Measure: Select Measurement Type

Title	Type	Property
Unique Devices	Unique	Device Id

Figure 13-9. Unique Devices chart definition

The following chart is created:

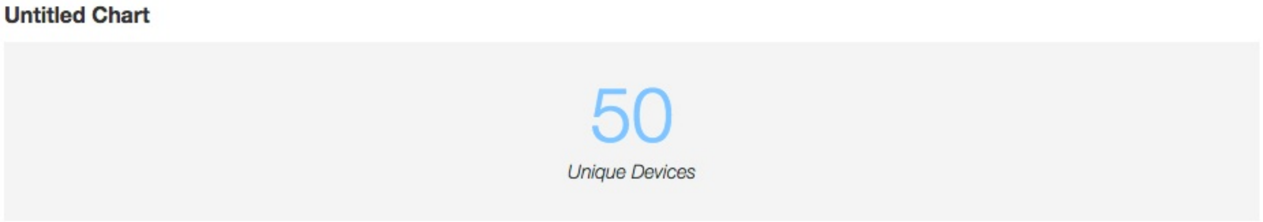


Figure 13-10. Unique Devices Metric Group Chart

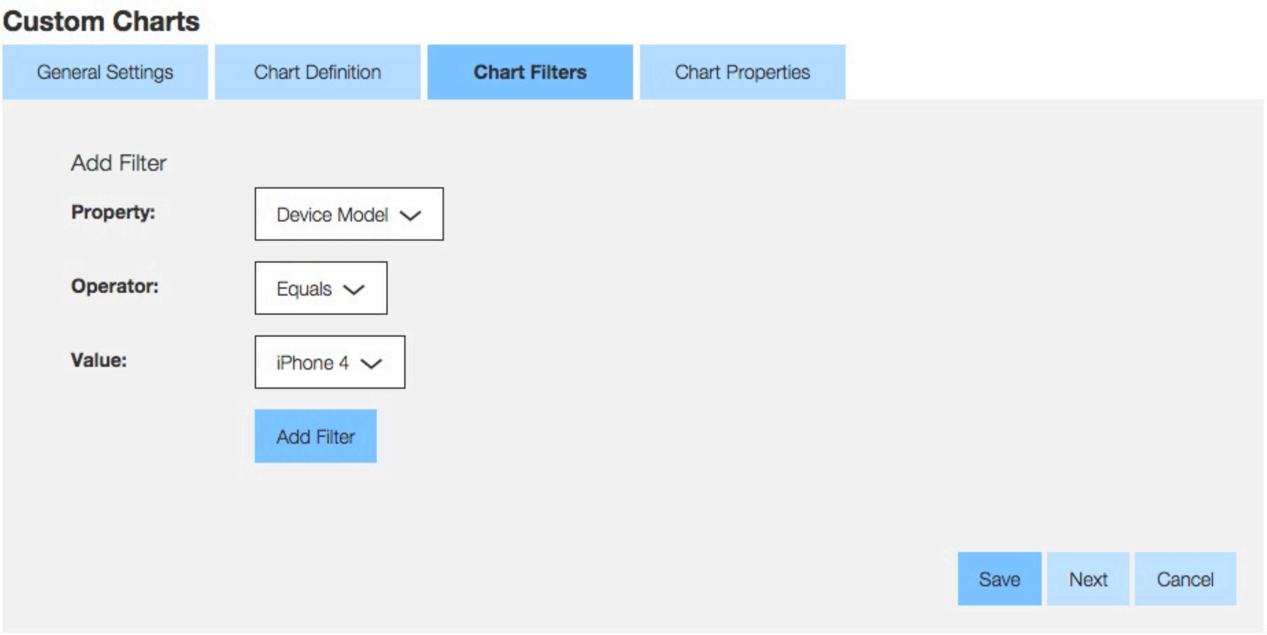
Filters:

When you create a custom chart, you can apply filters to the results from the **Filters** tab.

As discussed in “Event types” on page 13-10, each event type has a set of properties. You can apply the following filters for any of these properties:

- Equals/Not Equals
- Exists/Not Exists
- Greater Than/Greater Than or Equal
- Less Than/Less Than or Equal

You can create a bar graph chart to display the number of application activities for each day. You want to see the number of activities that were made by iPhone 4 devices. You can apply the **Equals** filter to your chart:



Property: Device Model

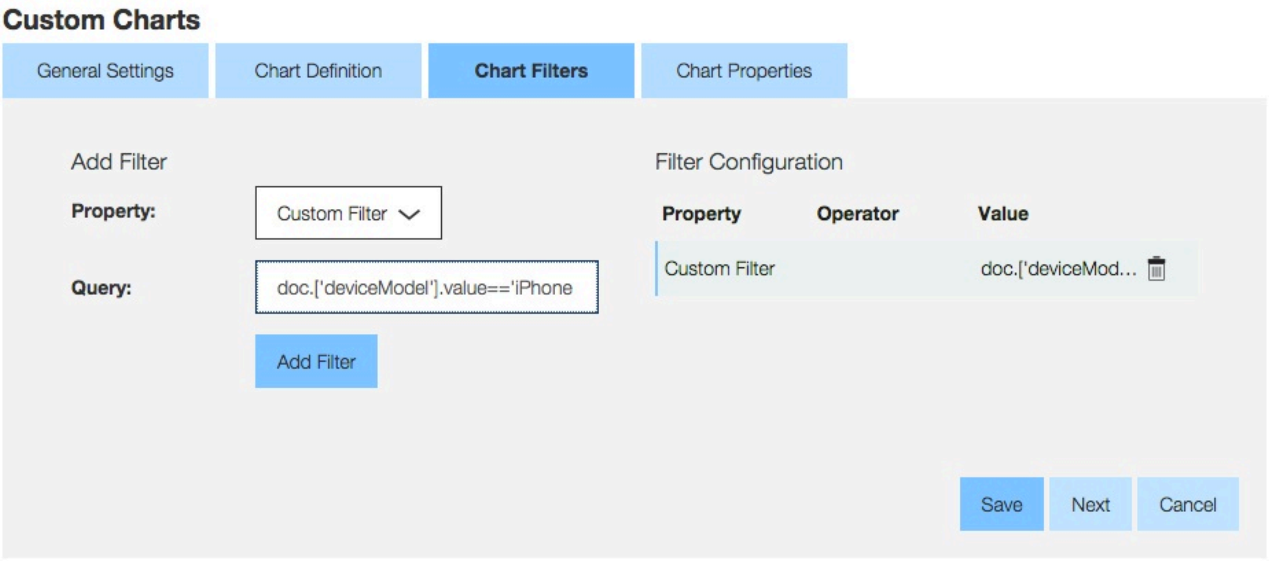
Operator: Equals

Value: iPhone 4

Figure 13-11. Device Model Filter

Custom Filters

Several filters can be applied to the same chart, but all of these filters are applied together. For example, if you want to see application activities for iPhone 4 and iPhone 5 devices, and you cannot add an Equals filter for both device model values. If you do, the resulting query searches for activities where the device model is iPhone 4 and iPhone 5. This query returns zero results since each activity can have only one device model property. To create more complex predicates for queries, you must use the custom filters feature.



Property: Custom Filter

Query: doc.['deviceModel'].value=='iPhone

Property	Operator	Value
Custom Filter		doc.['deviceMod... 🗑

Figure 13-12. Custom Filters

The following filter selects only the documents where the device model is iPhone 4:

```
doc['deviceModel'].value=='iPhone 4'
```

Conversely, this filter does the opposite and selects only the documents where the environment is not iPhone 4:

```
doc['deviceModel'].value!='iPhone 4'
```

You can do other logical comparisons, like the greater than and less than comparisons:

```
doc['timestamp'].value<1418277600000
```

In addition to logical expressions, you can combine filters by using Boolean expressions. You can filter on documents where the environment is Android or iPhone:

```
doc['deviceModel'].value=='iPhone 4' || doc['deviceModel'].value=='iPhone 5'
```

You can further group these filters by using parenthesis. You can combine the last filter with a limitation to retrieve all documents before a certain date:

```
doc['timestamp'].value<1418277600000 && (doc['deviceModel'].value=='iPhone 4' || doc['deviceModel'].value=='iPhone 5')
```

You can extract other data from a document field besides its value. For example, `doc['field_name'].values` is the native array values of the field, and `doc['field_name'].empty` is a Boolean indicating if the field has no values within the document. For a detailed description of all the documents and their fields, refer to. For a list of all available expressions, see Elastic Search scripting documentation

Note: When you create a custom filter for a custom chart, the query of a custom filter does not use Unicode Normalization Forms C (NFC) for comparison. The query does not consider language sensitivity such as normal matching, accent-insensitive, case-insensitive, and 1-to-2 mapping for search function to run correctly in different languages. The message property uses NFC and considers language sensitivity though. For fuzzy searches, only right like searches are compatible with the message query.

Patterns for creating custom charts:

Each of the following sections contains a subsection that explains how to populate data, which typically involves calling some APIs on the client.

For more information about the different types of data that is collected, see "Event types" on page 13-10.

Note: Some APIs, such as `connect` and `invokeProcedure` send analytics information automatically without having to use the analytics `send` API. However, this behavior is only true if you call `connect` and `invokeProcedure` 60 seconds apart. This implementation limits the number of requests to the server. For example, when you call multiple adapters in the span of a single minute, it is better to "batch send" the analytics information. For testing purposes, it is better to always call the analytics `send` API explicitly.

Client Logs:

Client logs contain log information sent with the platform's `Logger` API. This also includes contextual information about the device, including environment, application name, and application version.

Before you begin

If you are using the `WL.Client.logActivity` API, replace those calls with `WL.Analytics.log`. Otherwise you cannot create custom charts from the information you are logging.

Note: The steps that follow use JavaScript APIs. If you are using a native application, you must use the native versions of these APIs.

About this task

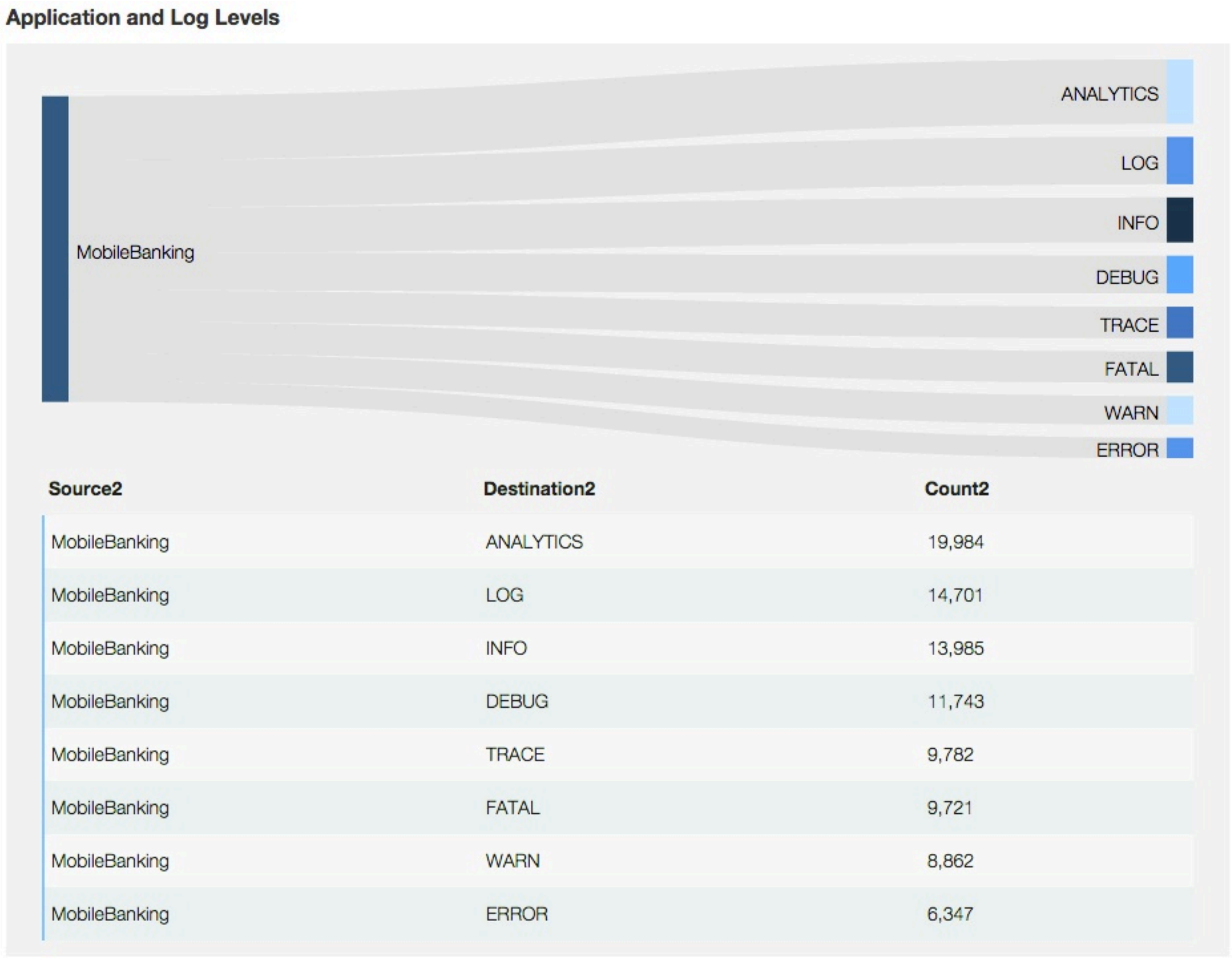
In this example, you use client log data to create a flow chart. The final graph shows the distribution of log levels in a specific application. You also have the following data available to show in a chart:

- Specific data
  - Log level
- Message data
  - Timestamp
- Device OS Contextual data
  - Application name
  - Application version
  - Device OS
- Device Contextual data
  - Device Id
  - Device model
  - Device OS version

Procedure

1. Create a new hybrid application.
2. Populate data by calling `WL.Logger.log('Hello world!')` or any of the other `Logger` APIs. Then call `WL.Logger.send()`.
3. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.
4. Click **Create Chart** to create a new custom chart.
5. Provide the following values:
   - **Chart Title:** Application and Log Levels
   - **Event Type:** Client Logs
   - **Chart Type:** Flow Chart
6. Click the **Chart Definition** tab.
7. Provide the following values:
   - **Source:** Application Name
   - **Destination:** Level
   - **Property:** your application name
8. Click **Save**.

Results



Application and Log Levels

Source2	Destination2	Count2
MobileBanking	ANALYTICS	19,984
MobileBanking	LOG	14,701
MobileBanking	INFO	13,985
MobileBanking	DEBUG	11,743
MobileBanking	TRACE	9,782
MobileBanking	FATAL	9,721
MobileBanking	WARN	8,862
MobileBanking	ERROR	6,347

Figure 13-13. Application and Log Levels

Server Network Transactions:

Server network activities contain information about network requests such as network request time and bytes received by the client. This event type also includes contextual information about the device, such as environment, application name, and application version.

About this task

In this example, you use server network activity data to create a line chart. The graph shows the average bytes that are received. All the values in a single day are averaged. You also have the following data available to show in a chart:

- Specific data
  - Adapter name
  - Procedure name
  - Adapter response time (in milliseconds)
  - Bytes received
  - Server (for example: `dallas.ibm.com/9.64.77.98`)

  - URL path (for example: /wlproj/apps/services/api/wlapp/iphone/query)
  - Authenticator Name
  - Success status
  - Login Module Name
  - Login Module Class
  - Tracking ID
  - Realm Name
  - Session Id
  - Validation Code
  - Client Id
- Message data
  - Timestamp
- Device OS Contextual data
  - Application name
  - Application version
  - Device OS
- Device Contextual data
  - Device Id
  - Device model
  - Device OS version

Procedure

1. Create a new hybrid application.
2. Populate data.
   a. Call WL.Client.connect().
   b. Create an adapter with a procedure.
      - helloWorldAdapter-impl.js

        ```
        function sayHello() {
          return {hello: 'world'};
        }
        ```

      - helloWorldAdapter.xml

        ```
        <procedure name="sayHello"/>
        ```

   c. Call WL.Client.invokeProcedure() to call a procedure on the newly created adapter. The parameters include anything that is required to contact your adapter. The following example uses an adapter that returns a simple {hello: 'world'} object.
      - main.js

        ```
        WL.Client.invokeProcedure({
          adapter : 'helloWorldAdapter',
          procedure : 'sayHello',
          parameters : []
        })
        .then(function (res) {
          console.log(JSON.stringify(res, null, ' '));
        })
        ```

   d. Call WL.Analytics.send().
3. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.

4. Click **Create Chart** to create a new custom chart.
5. Provide the following values:
   - **Chart Title:** Average Bytes Received
   - **Event Type:** Server Network Activities
   - **Chart Type:** Line Graph
6. Click the **Chart Definition** tab.
7. Provide the following values:
   - **Measure:** Average
   - **Property:** Bytes Received
8. Click **Save**.

Results

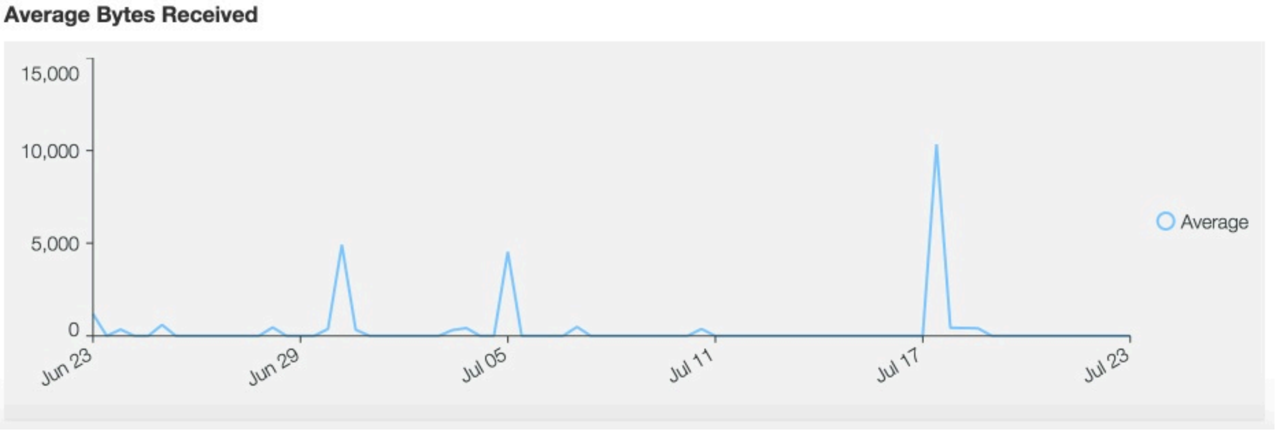


Figure 13-14. Average Bytes Received line graph

Push Notifications:

Push notifications contain information about push notifications.

About this task

In this example, you use the notification activities information to create a table. The table shows an application name and the mediator (for example, APNS is Apple Push Notification Service). You also have the following data available to show in a chart:

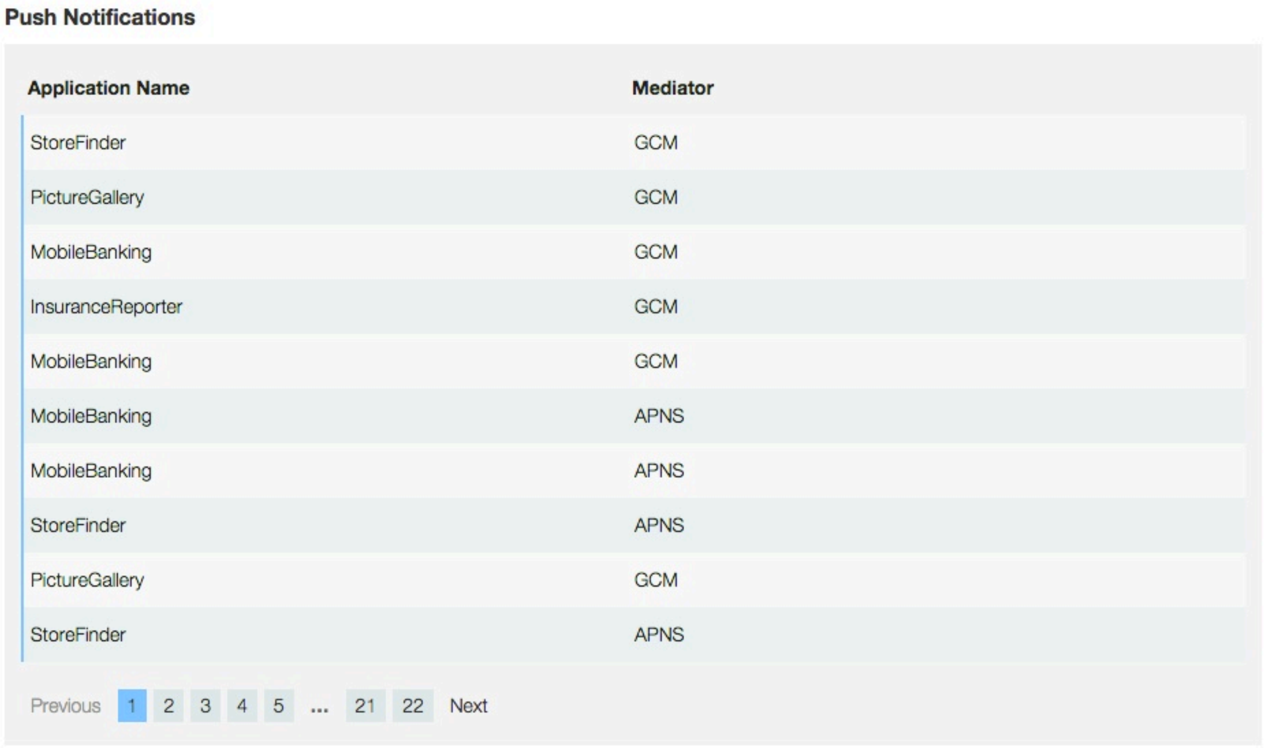
- Event source
- Application version
- Mediator
- Application name

Procedure

1. Create a new hybrid application.
2. Set up push notifications. For more information, see "Setting up push notifications" on page 8-172.
3. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.

4. Click **Create Chart** to create a new custom chart.
5. Provide the following values:
   - **Chart Title:** Push Notifications
   - **Event Type:** Push Notifications
   - **Chart Type:** Table
6. Click the **Chart Definition** tab.
7. Add the following columns: `Application Name` and `Mediator`
8. Click **Save**.

Results



Push Notifications

Application Name	Mediator
StoreFinder	GCM
PictureGallery	GCM
MobileBanking	GCM
InsuranceReporter	GCM
MobileBanking	GCM
MobileBanking	APNS
MobileBanking	APNS
StoreFinder	APNS
PictureGallery	GCM
StoreFinder	APNS

Figure 13-15. Table of Push Notifications

Server Logs:

Server logs contain internal server logs.

About this task

In this example, you use the server log information to create a metric group. The metric shows the number of log messages on the server that are categorized as debug by their corresponding log level property. You also have the following data available to show in a chart:

- Specific data
  - Log level
  - Message
  - Logger name
  - Source class

  - Server (for example: `dallas.ibm.com/9.64.77.98`)
  - Source
  - Source method
  - Thread Id
  - URL path (for example: `/wlproj/apps/services/api/wlapp/iphone/query`)
- Message data
  - Timestamp

Procedure

1. Create a new hybrid application.
2. Populate data
   a. Call `WL.Client.connect()`.
   b. Create an adapter with a procedure.
      - `helloWorldAdapter-impl.js`

```
function sayHello() {
  return {hello: 'world'};
}
```

      - `helloWorldAdapter.xml`

```
<procedure name="sayHello"/>
```

   c. Call `WL.Client.invokeProcedure()` to call a procedure on the newly created adapter. The parameters include anything that is required to contact your adapter. The following example uses an adapter that returns a simple `{hello: 'world'}` object when called from the client.
      - main.js

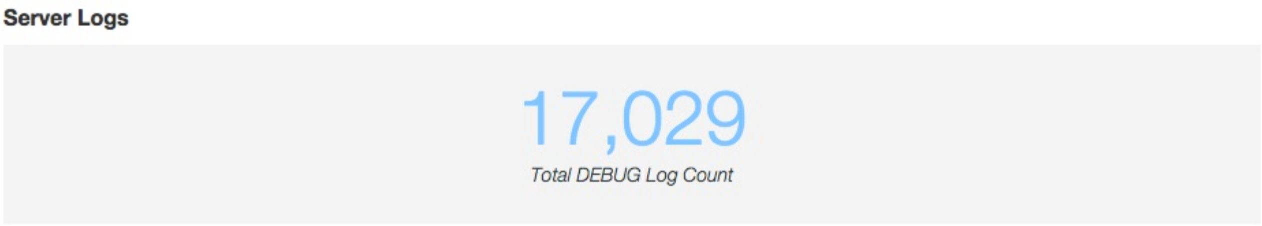
```
WL.Client.invokeProcedure({
    adapter : 'helloWorldAdapter',
    procedure : 'sayHello',
    parameters : []
})
.then(function (res) {
    console.log(JSON.stringify(res, null, ' '));
})
```

   d. Call `WL.Analytics.send()`.
3. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.
4. Click **Create Chart** to create a new custom chart.
5. Provide the following values:
   - **Chart Title:** Server Logs
   - **Event Type:** Server Logs
   - **Chart Type:** Metric Group
6. Click the **Chart Definition** tab.
7. Provide the following values:
   - **Metric Title:** Total DEBUG Log Count
   - **Measure:** Total
8. Click **Add Metric**.
9. Click the **Chart Filters** tab and select the following options:
   - **Filter:** Level
   - **Operator:** Equals

- **Value:** DEBUG

10. Click **Add Filter**.
11. Click **Save**.

Results


Server Logs

17,029

Total DEBUG Log Count

Figure 13-16. Total Debug Log Count

Creating a custom chart with custom data:

To collect custom data, configure your application codebase with the Analytics API.

About this task

In this scenario, you create a pie chart that represents the relative frequency of user button clicks.

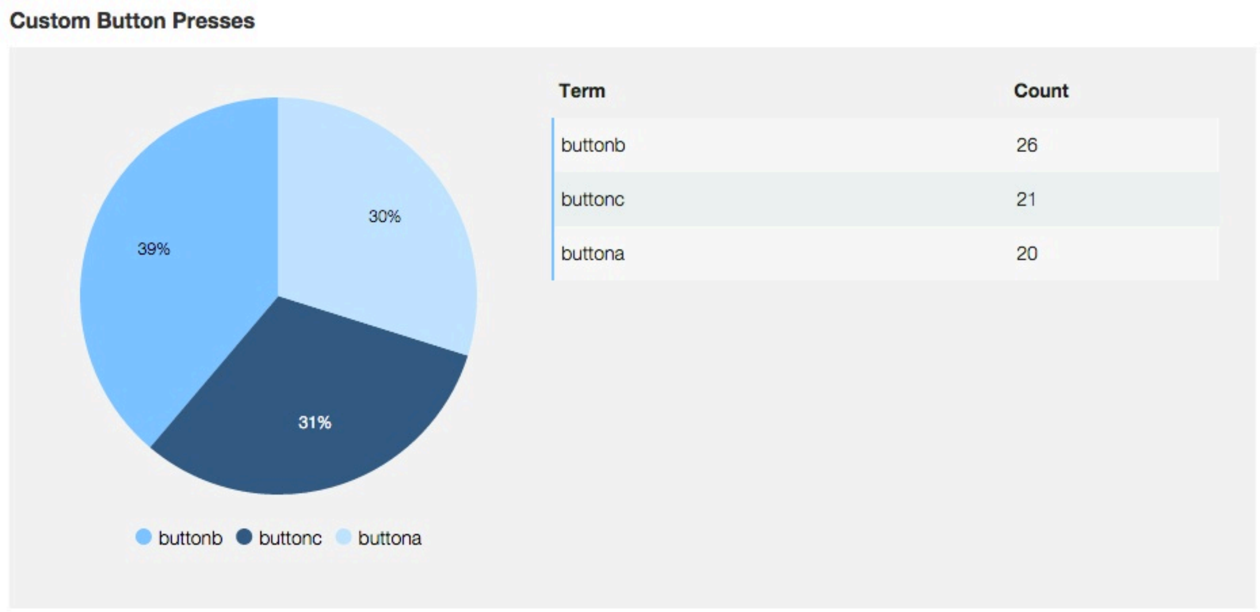
Procedure

1. Create a MobileFirst application. For this scenario, attach functions to the JavaScript `onClick` event listeners for buttons in your user interface. These functions are instrumented to report analytics. The `WL.Analytics.log` method takes an object and a message as parameters. The message is used for searching for this custom event in the **Search** page of the Analytics Console. The object is used to collect data for custom analytics charting.

```
function buttonA(){
  var event = {buttonPress: 'buttona'};
  WL.Analytics.log(event, 'Custom event for button A press');
}

function buttonB(){
  var event = {buttonPress: 'buttonb'};
  WL.Analytics.log(event, 'Custom event for button B press');
}

function buttonC(){
  var event = {buttonPress: 'buttonc'};
  WL.Analytics.log(event, 'Custom event for button C press');
}

function sendAnalytics(){
  WL.Analytics.send();
}
```

   The `sendAnalytics` function is a convenience for demonstration purposes only. The values for keys that are passed to the `WL.Analytics.log` method can be strings and numbers only. Objects and arrays can be passed as values, but are not used during custom charts creation. The `event` object in each method is used to create custom charts. The `buttonPress` key is used to generate data for each chart. The values for this key are `buttona`, `buttonb`, and `buttonc`.

2. Create a custom chart. For more information about custom charts, see “Creating a custom chart” on page 13-38.
   When a device sends the analytics to the server, the key becomes available in the **Select Property** field for chart creation and filters.



Term	Count
buttonb	26
buttonc	21
buttona	20

Figure 13-17. Chart Definition

Patterns for visualizing custom data:

You can instrument your IBM MobileFirst Platform Foundation for iOS V7.1.0 applications to collect and visualize analytics data, so you can continuously understand application usage and target improvements.

Prerequisites

Ensure that you are familiar with the `WL.Analytics` API and “Operational analytics” on page 13-6.

Patterns

When you develop mobile applications, you can track key actions over time that can provide useful insight. The following patterns explain how to track various interesting data points and chart them in different ways.

Many visualizations can be created following two patterns. The first pattern involves recording and sending information as it gets created. For example, a user completes a registration form. The second pattern involves accumulating and calculating information on the client-side before you call the analytics API to record and send that information to the server. For example, the amount of time that is elapsed between two actions.

Single events:

The single event pattern involves sending a single piece of information as it becomes available.

One example of a single event pattern is tracking new user registrations. After the user successfully completed the application's registration form, tracking this data gives insight about how many people are registering for your application over time.

Creating a line graph chart for new user registrations over time:

You can create a line graph chart to represent new registered users over time.

Procedure

1. Create a MobileFirst hybrid application. These instructions use the JavaScript versions of the `log` and `send` APIs. If you want to work with a native application instead, use the native versions of `log` and `send`.
2. Write the following code to generate four `newSignup` events:

   ```
   WL.Analytics.log({ newSignup: 'username' + Math.random(), _activity: "customActivity" }, 'newSign
   WL.Analytics.log({ newSignup: 'username' + Math.random(), _activity: "customActivity" }, 'newSign
   WL.Analytics.log({ newSignup: 'username' + Math.random(), _activity: "customActivity" }, 'newSign
   WL.Analytics.log({ newSignup: 'username' + Math.random(), _activity: "customActivity" }, 'newSign
   ```

   This code locally stores a JSON object with information about the new registrations, which is a few random user names in this case. The second parameter in the `log` function is a useful string to search for log messages. However, for the creation of custom charts, this parameter is irrelevant. The example uses the same key as the object that is being recorded.
3. Send analytics data to the server with the following function:

   ```
   WL.Analytics.send();
   ```

   This function detects all analytics messages that are locally stored, sends them to the server, and removes the local copy after successfully sending them.
4. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.
5. Click **Create Chart** to create a new custom chart.
6. Provide the following values:
   - **Chart Title:** New Registrations
   - **Event Type:** Custom Data
   - **Chart Type:** Line Graph
7. Click the **Chart Definition** tab.
8. Provide the following values:
   - **Measure:** Unique
   - **Property:** newSignup
9. Click **Save**.

Results

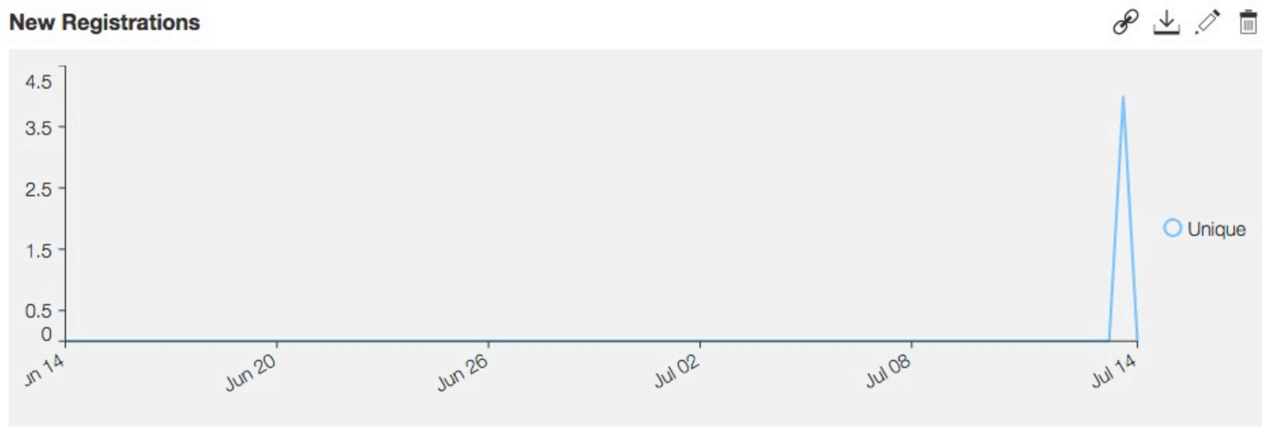


Figure 13-18. New User Registrations Over Time

Creating a table chart for new registrations and device OS:

You can create a table chart to represent new registered users and what device OS they are using.

Before you begin

Complete steps 1-5 in "Creating a line graph chart for new user registrations over time" on page 13-56.

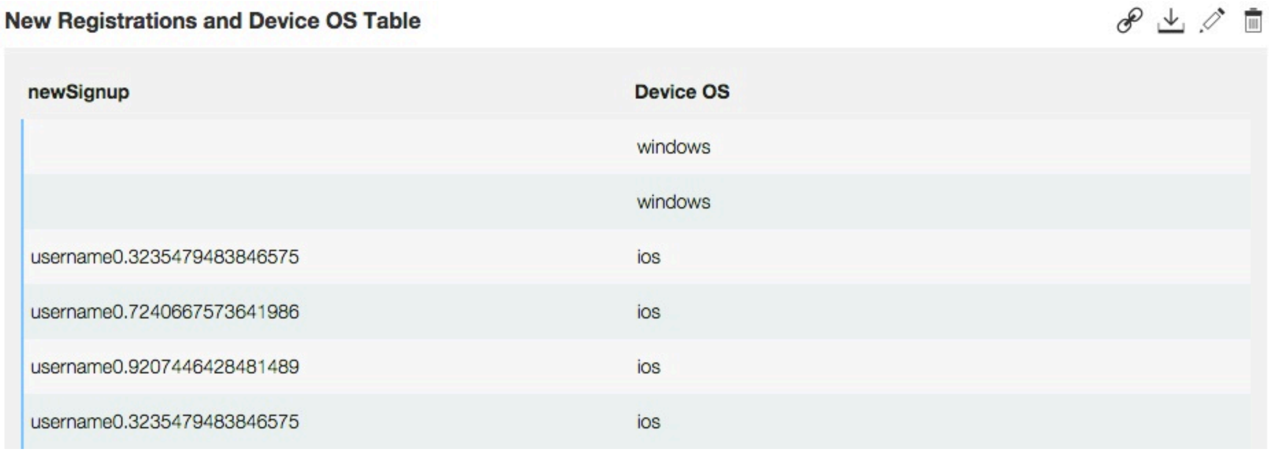
About this task

In this example, you create a table of new registered users and what device OS they are using.

Procedure

1. Provide the following values when you add the new chart:
   - **Chart Title:** New Registrations and Device OS Table
   - **Event Type:** Custom Data
   - **Chart Type:** Table
2. Click the **Chart Definition** tab.
3. Add the following columns to the table: `newSignup` and `Device OS`
4. Click **Save**.

Results



New Registrations and Device OS Table

newSignup	Device OS
	windows
	windows
username0.3235479483846575	ios
username0.7240667573641986	ios
username0.9207446428481489	ios
username0.3235479483846575	ios

Figure 13-19. New Registrations and Device OS Table

What to do next

You can export the data that is shown in the table to use with other applications. For more information about exporting data, see “Exporting custom data” on page 13-65.

Creating a metric group chart for new registrations:

You can create a metric group chart to represent new user registration data.

Before you begin

Complete steps 1-5 in “Creating a line graph chart for new user registrations over time” on page 13-56.

About this task

In this example, you create a metric group chart of new registrations.

Procedure

1. Provide the following values when you add the new chart:
   - **Chart Title:** New Registrations Metric Group
   - **Event Type:** Custom Data
   - **Chart Type:** Metric Group
2. Click the **Chart Definition** tab.
3. Add the following Metric information:
   - **Metric Title:** Registrations
   - **Metric Type:** Unique
   - **Metric Property:** newSignup
4. Click **Add Metric**.
5. Click **Save New Chart**.

Results

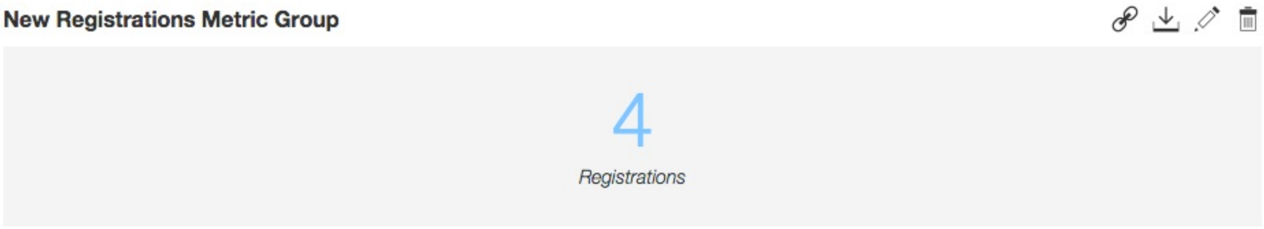


Figure 13-20. New Registrations Metric Group

Creating a pie chart for check-out results:

You can create a pie chart to represent check-out successes and failures.

About this task

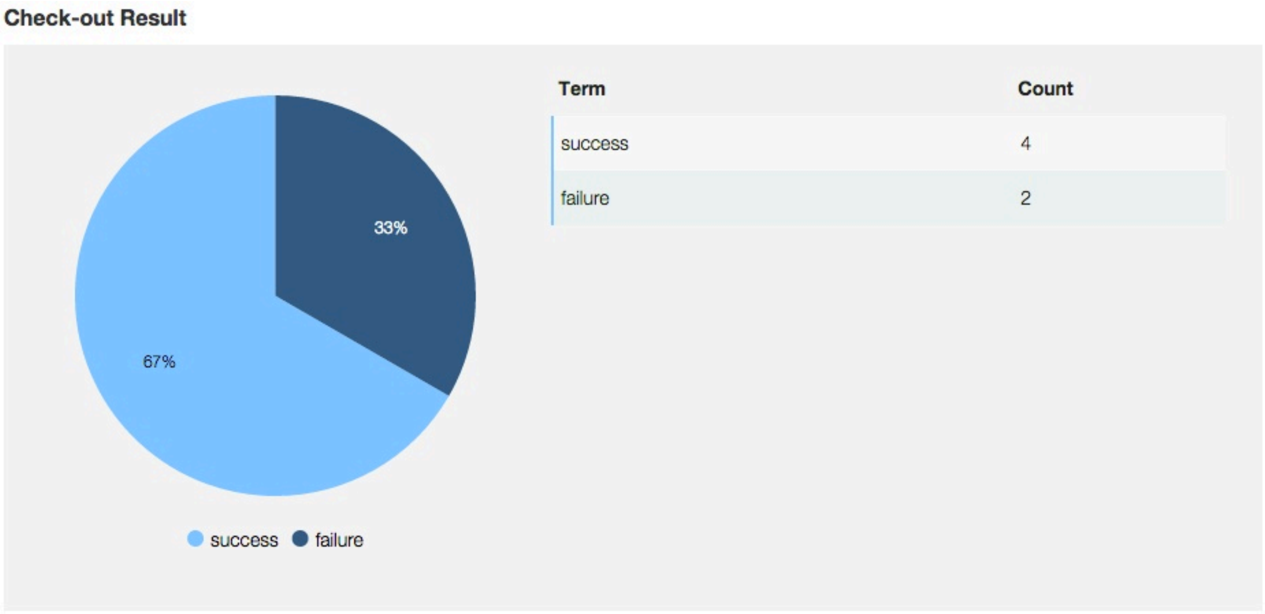
You can visualize data as a pie chart. This example shows the percentage of check-out successes and failures.

Procedure

1. Create a MobileFirst hybrid application.
2. Write the following code to generate four successful check-out events and two failures:

   ```
   WL.Analytics.log({checkout: 'success', _activity: "customActivity"}, 'checkout');
   WL.Analytics.log({checkout: 'success', _activity: "customActivity"}, 'checkout');
   WL.Analytics.log({checkout: 'success', _activity: "customActivity"}, 'checkout');
   WL.Analytics.log({checkout: 'success', _activity: "customActivity"}, 'checkout');
   WL.Analytics.log({checkout: 'failure', _activity: "customActivity"}, 'checkout');
   WL.Analytics.log({checkout: 'failure', _activity: "customActivity"}, 'checkout');
   ```
3. Provide the following values when you add the new chart:
   - **Chart Title:** Check-out Result
   - **Event Type:** Custom Data
   - **Chart Type:** Pie Chart
4. Click the **Chart Definition** tab.
5. Provide the following values:
   - **Property:** checkout
6. Click **Save**.

Results


Check-out Result

Term	Count
success	4
failure	2

Figure 13-21. Check-out Results

Composed events:

Some pieces of information are only meaningful after you do certain calculations on them.

Examples of composed events are tracking the length of a user session and how long it took a user to complete a specific task. The client portion of the application must record an initial value, such as the current date, wait for the user to complete the task of interest, and then record the amount of time that passed between the initial value and the final value. The client then sends the result of that computation to the server.

The key difference between composed events and single events is that you cannot log `sessionStartTime` and `sessionEndTime` as they occur and have the server calculate the time that is spent between the two intervals to generate a meaningful chart.

In addition to time-based information, another use case for composed events is recording the source and destination page of a user's navigation through the application.

You can also record information about the user and augment analytics events that are sent to the server. For example, recording the users that used the application more than a certain number of times are considered active users. You can send information about the session, such as what country the user is in, and then filter by active users.

Session Length:

You can plot average session lengths and define a threshold to visualize how often the sessions lengths exceed your threshold.

About this task

In this example, you take all the values that are recorded in a single day, calculate the average, and plot them on a graph.

Procedure

1. Define what you consider a session. A session can be defined as the time interval that starts when the user logs in until the user logs out. Another way to define a session is to start the session when the user begins a process and end the session when the process is complete. For this example, a session starts when the application enters the foreground and ends when the application goes into the background.
2. The application lifecycle callback methods depend on which environments you want to support. You can use the native versions of Analytics API for those environments. For simplicity, this example logs the amount of time that is spent on the session that uses the JavaScript API.
3. Add this code to your application:

   ```
   function getSessionTotalTime(currentTime, startTime) {
     return currentTime - startTime;
   }

   // startTime and endTime can be retrieved through app lifecycle event listeners

   WL.Analytics.log({'totalSessionTime' : getSessionTotalTime(endTime, startTime)}, 'totalSession
   ```
4. Send the analytics data to the server:

   ```
   WL.Analytics.send();
   ```
5. From the **Custom Charts** tab of the Dashboard page in the Analytics Console, click **Create Chart**.
6. Provide the following values:
   - **Chart Title:** Average Session Times
   - **Event Type:** Custom Data
   - **Chart Type:** Bar Graph
7. Click the **Chart Definition** tab and provide the following values:
   - **X-Axis:** Axis By Timeline
   - **Y-Axis:** Average
   - **Property:** totalSessionTime
8. Click the **Chart Properties** tab and add a threshold line with the following values:
   - **Threshold Label:** SessionTimeGoal
   - **Threshold Value:** 300

   This threshold creates a line at the value that is specified with the label. The goal is to show the daily average session time that is greater than 300 units of time.
9. Click **Save**.

Results

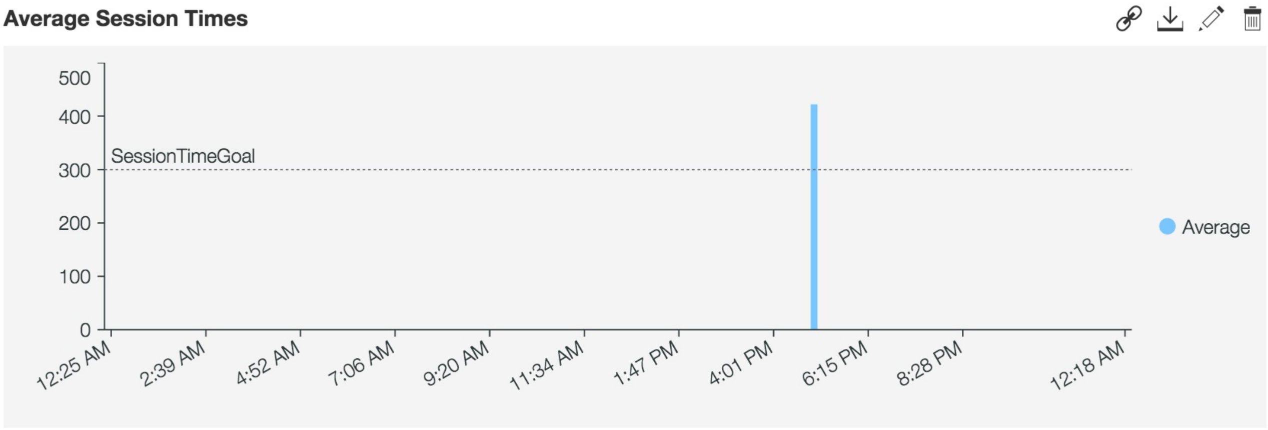


Figure 13-22. Session Length

Page Flow:

You can gain insight about how users of your application reach and leave certain pages by tracking the source page and the destination page.

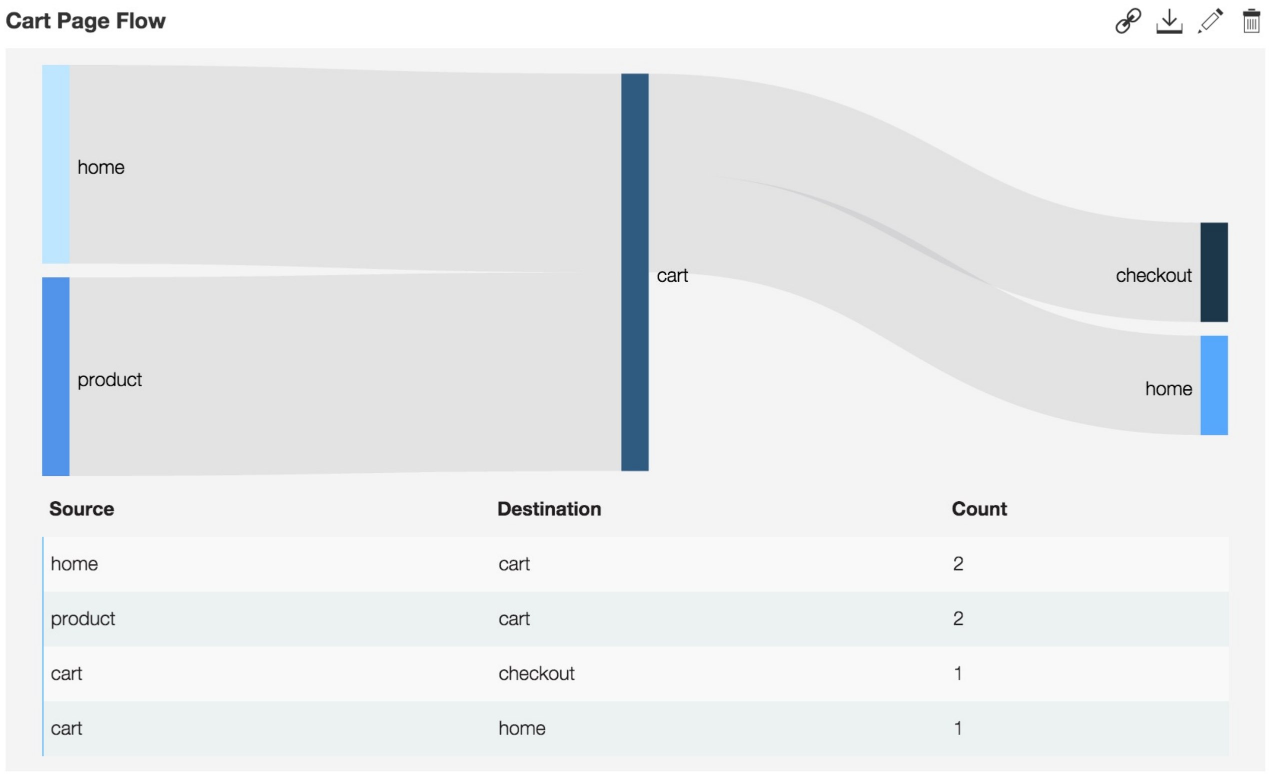
About this task

In this example, you record how users reach the `cart` page of an application, and what pages users moves to after they leave that page.

Procedure

1. Create a new hybrid application.
2. Make the following calls to the log function.
```
WL.Analytics.log({sourcePage : 'home', destinationPage: 'cart'}, 'pageFlow');
WL.Analytics.log({sourcePage : 'home', destinationPage: 'cart'}, 'pageFlow');
WL.Analytics.log({sourcePage : 'product', destinationPage: 'cart'}, 'pageFlow');
WL.Analytics.log({sourcePage : 'product', destinationPage: 'cart'}, 'pageFlow');
WL.Analytics.log({sourcePage : 'cart', destinationPage: 'home'}, 'pageFlow');
WL.Analytics.log({sourcePage : 'cart', destinationPage: 'checkout'}, 'pageFlow');
```
3. In the MobileFirst Analytics Console, click the **Custom Charts** tab on the Dashboard page. You can create a chart based on the analytics messages that were sent to the server.
4. Click **Create Chart** to create a new custom chart.
5. Provide the following values:
   - **Chart Title:** Cart Page Flow
   - **Event Type:** Custom Data
   - **Chart Type:** Flow Chart
6. Click the **Chart Definition** tab.
7. Provide the following values:
   - **Source:** sourcePage
   - **Destination:** destinationPage
   - **Property:** cart
8. Click **Save**.

Results



Cart Page Flow

Source	Destination	Count
home	cart	2
product	cart	2
cart	checkout	1
cart	home	1

Figure 13-23. Cart Page Flow

Regular Users and Geolocation:

You can view the geographical locations for regular users of your application.

About this task

In this example, you show the geographical locations of an application's regular users.

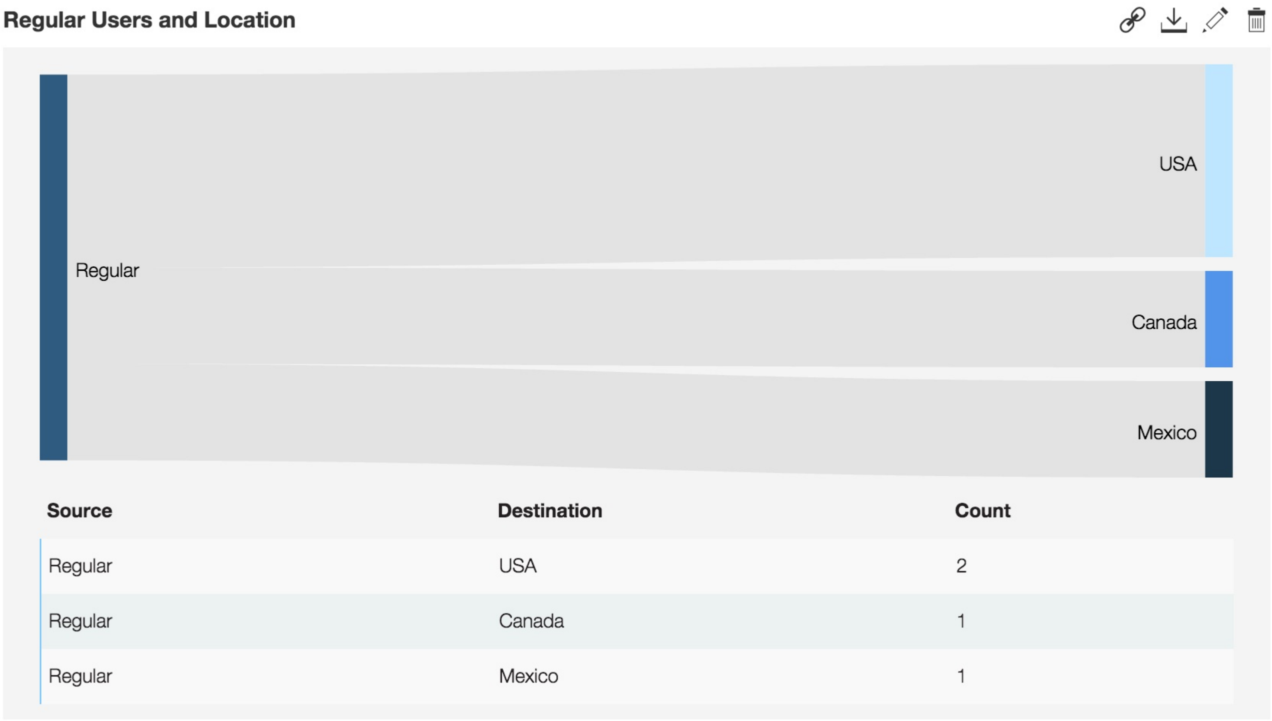
Procedure

1. Define different types of users that are relevant to your application. For this example, use the following types of users:
   - New: The user used the application less than 5 times.
   - Regular: The user used the application 5 or more times.
2. When the application is started, you need to persist a counter with how many times the user opened the application. This example uses HTML5 local storage, but you can use something more powerful, such as JSONStore. Use the following code to obtain the type of user:
```
var userVisit = localStorage.getItem('userVisit');
localStorage.setItem('userVisit', userVisit + 1);
var typeOfUser = (userVisit > 5) ? 'Regular' : 'New';
```
3. Send the following log messages:
```
WL.Analytics.log({type : 'New', geographicLocation: 'USA'}, 'visit');
WL.Analytics.log({type : 'New', geographicLocation: 'Mexico'}, 'visit');
WL.Analytics.log({type : 'New', geographicLocation: 'USA'}, 'visit');
WL.Analytics.log({type : 'Regular', geographicLocation: 'Mexico'}, 'visit');
WL.Analytics.log({type : 'Regular', geographicLocation: 'Canada'}, 'visit');
WL.Analytics.log({type : 'Regular', geographicLocation: 'USA'}, 'visit');
WL.Analytics.log({type : 'Regular', geographicLocation: 'USA'}, 'visit');
```

In a real application, do not hardcode values for `type` and `geographicLocation`. When you get the location, use strings instead of latitude longitude values.

4. From the **Custom Charts** tab of the Dashboard page in the Analytics Console, click **Create Chart**.
5. Provide the following values:
   - **Chart Title:** Regular Users and Location
   - **Event Type:** Custom Data
   - **Chart Type:** Flow Chart
6. Click the **Chart Definition** tab and provide the following values:
   - **Source:** type
   - **Destination:** geographicLocation
   - **Property:** Regular
7. Click **Save**.

Results



Source	Destination	Count
Regular	USA	2
Regular	Canada	1
Regular	Mexico	1

Figure 13-24. Regular Users and Location

New Users and Location:

You can use filters with a pie chart to show new users and their geographic location instead of regular users.

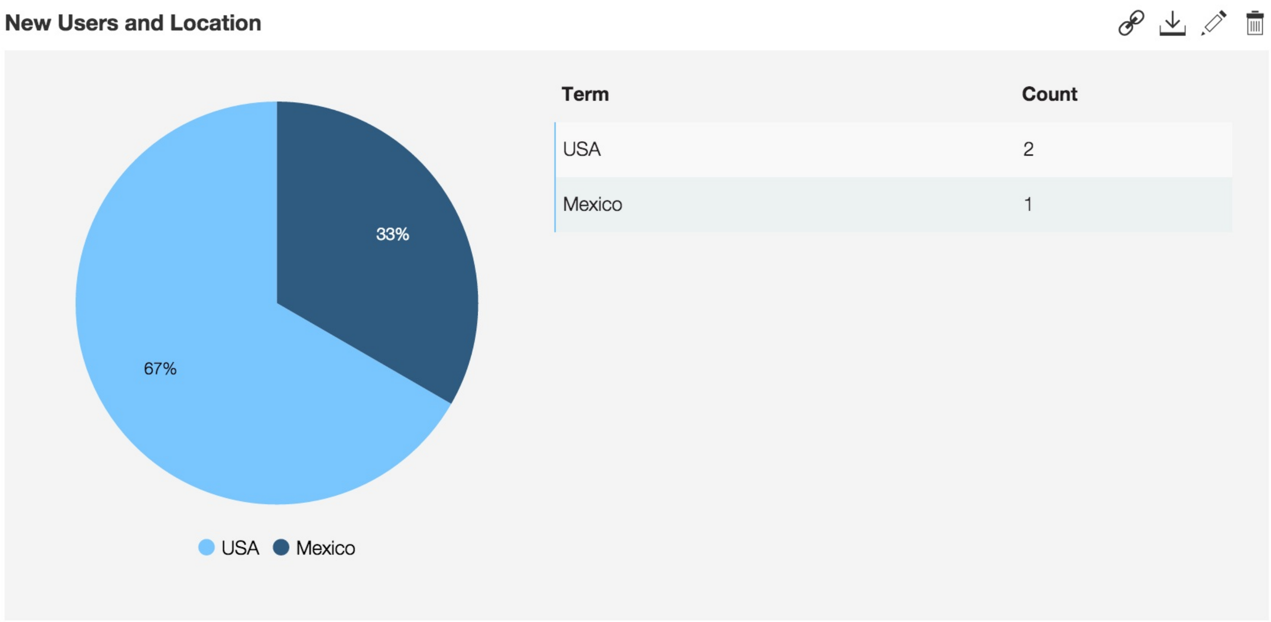
About this task

In this example, the chart uses the same information that is presented in “Regular Users and Geolocation” on page 13-63, but shows new users instead of regular users. Unlike that task, this example uses filters to limit the amount of information that is shown and uses a pie chart instead of a flow chart.

Procedure

1. Follow steps 1-3 from “Regular Users and Geolocation” on page 13-63.
2. From the **Custom Charts** tab of the Dashboard page in the Analytics Console, click **Create Chart**.
3. Provide the following values:
   - **Chart Title:** New Users and Location
   - **Event Type:** Custom Data
   - **Chart Type:** Pie Chart
4. Click the **Chart Definition** tab and provide the following values:
   - **Property:** geographicLocation
5. Click the **Chart Filters** tab and provide the following values:
   - **Property:** type
   - **Operator:** Equals
   - **Value:** New
6. Click **Save**.

Results



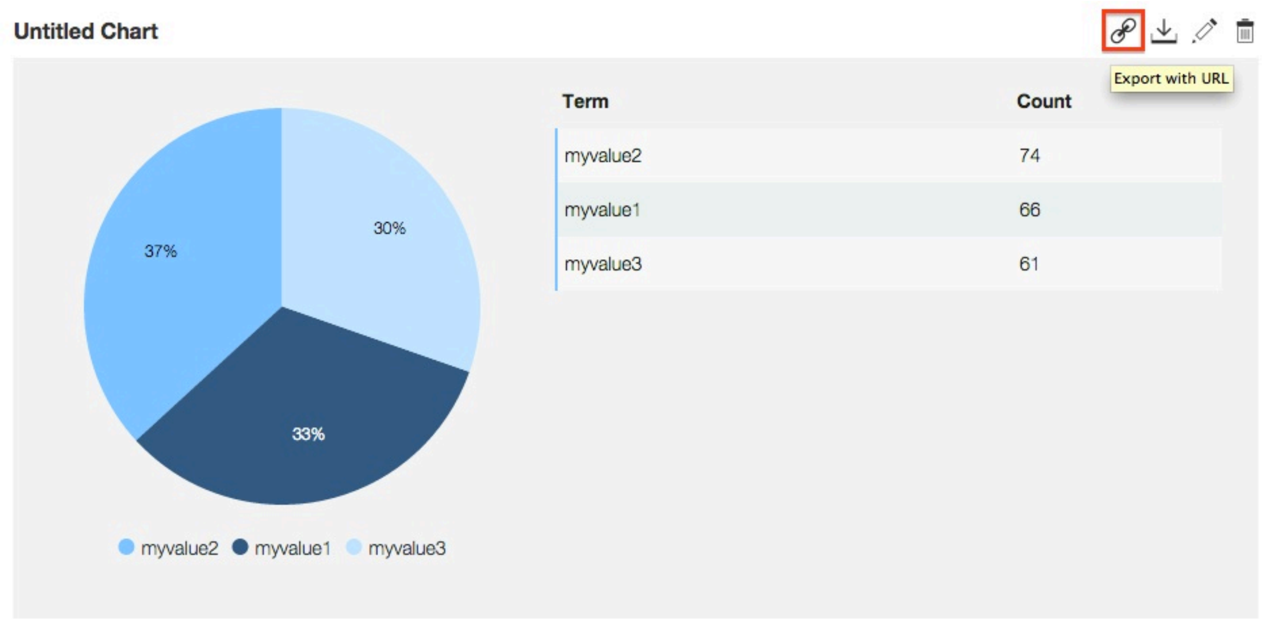
Term	Count
USA	2
Mexico	1

Figure 13-25. New Users and Location

Exporting custom data:

The data from each custom chart can be exported into JSON, XML, or CSV format.

The structure of the exported data depends on the chart that is being exported. To export data, click the export icon at the upper right of the custom chart.



Term	Count
myvalue2	74
myvalue1	66
myvalue3	61

Figure 13-26. Exporting chart data

Exporting and importing custom chart definitions:

If you are using the latest interim fix of MobileFirst, you can import and export custom chart definitions programmatically or manually in the IBM MobileFirst Platform Operational Analytics Console.

Before you begin

Ensure that you have at least one custom chart in the IBM MobileFirst Analytics Console.

About this task

In this example, you manually export and import custom chart definitions.

Procedure

1. In the MobileFirst Analytics Console, click the **Custom Charts** tab in the Dashboard page.
2. To export the custom chart definitions, click **Export Charts**. This action displays a dialog to save a `customChartsDefinition.json` file.
3. Choose a location to save the file.
4. Click the **Delete Chart** icon next to each custom chart to delete all custom charts.
5. To import a custom chart definition, click **Import Charts**. This action displays a dialog to choose a file.
6. Choose the `customChartsDefinition.json` file that you previously exported to open.

Results

You exported and imported a custom chart definition manually in the MobileFirst Analytics Console.

What to do next

You can also export and import custom chart definitions programmatically by using your HTTP client of choice (for example, CURL or postman). The GET endpoint for export is `http://<hostname>:<port>/analytics-service/data/customCharts/apps/{appId}`. The POST endpoint for import is `http://<hostname>:<port>/analytics-service/data/customCharts/apps/{appid}/import`. For example, if the context root was not changed from the default of `analytics-service`, the export endpoint is `http://myHost.com:9080/analytics-service/data/customCharts/apps/{appid}` and the import endpoint is `http://myHost.com:9080/analytics-service/data/customCharts/apps/{appid}/import`. If the application is protected by basic authorization, all security requirements apply.

Note: If you import a custom chart definition that exists, you end up with duplicate definitions, which also means that the MobileFirst Analytics Console shows duplicate custom charts.

Security Analytics

Starting with V7.0.0, the MobileFirst Operational Analytics includes a **Security** tab.

Overview of IBM MobileFirst Platform Foundation for iOS security

Applications and adapter procedures can be protected by a security test. The test includes one or more realms. Each Realm represents a specific security check. Some realms are already provided. For example, `wl_antiXSRFRealm` is used to avoid cross-site request forgery attacks. Other realms are defined by the developer based on their security needs.

The realms contain two key pieces of functionality: an Authenticator and a Login Module. The Authenticator gets some information from the user or device, such as a user name and password. The Login Module gets information from the Authenticator and validates it. An example of validation is checking the user name and password against a user registry.

For more information about security, see “MobileFirst security overview” on page 11-87.

Example use case

1. A developer creates an adapter procedure. The procedure returns account information for a specific user.
2. Only valid users can access account information, so the developer protects the procedure with a security test. That adapter procedure is now a protected resource.
3. You can determine if a user is valid by contacting a user registry. The developer creates a realm and makes the authenticator the `FormBasedAuthenticator`. That authenticator will send a login form to the client and expect a response with the user name and password.

4. The Login Module is a Java class implemented by the developer. It contains code to contact a user registry on the backend using the credentials gathered by the `FormBasedAuthenticator`. There are two possible outcomes now:
   a. If the user is valid, the security test passes and access to the protected resource (account information for the user) is granted.
   b. If the security test fails, then the user is unable to access the protected resource. The cause of the failure might be incorrect credentials, failure to access the user registry due to a network error, exception on the server, or other causes.
5. If the user is valid, the security test passes and access to the protected resource (i.e. account information for the user) is granted. b) If the security test fails then the user is unable to access the protected resource. The cause of the failure might be one of the following: wrong credentials, failure to access the user registry due to a network error, exception on the server, among others.

Potential use cases

The platform can be used to develop a banking application. The application allows users to log in using their bank's credentials. Successfully authenticated users can access their account information and perform various actions (such as transferring money between accounts). This authentication flow is similar to the flow described in the previous section.

Network issues

If an application gets popular, more people are trying to access a protected resource (such as their account information). Using the **Security** tab of the Operational Analytics console, you notice a high failure rate on one of the realms. That realm is part of the security test used to allow users to log in and view their account information. The most common error under **Authenticator Failures** is an internal server error. One reason for this error is that the backend infrastructure (the number of servers) needs improvement to meet the increase in demand.

Security threats

You notice that there is a high failure rate on one of the protected resources (such as account information). Looking at the **Authenticator Failures**, you notice the most common error is invalid credentials. That error gets returned when the client supplies an invalid user name or password. This might mean a hacker is attempting to access a protected resource by trying different credentials within short time intervals. Based on this information, you conclude that you should not allow users to try different credentials indefinitely. With the data provided by the analytics console, you decide to let users attempt 5 wrong credentials before asking users to wait a specific amount of time before allowing them to try again. The goal is to limit the effectiveness of a brute force attack.

Application usability issues

In addition to using their user name and password to access the application and view their account balances, users also need to supply their bank's pin number to transfer money between accounts. You notice a high failure rate in the realm that asks for the pin number. Looking at the **Authenticator Failures**, you notice the most common error is client interaction required. This means the server is expecting credentials (the pin number) but is not receiving them. This can be caused by a high percentage of users inability to understand the user interface. The users are not noticing that they need to send their pin number. Based on these findings, the page should be redesigned to solve that issue.

Development defects

You are viewing security information from the last 24 hours and notice the failure rate for one of the realms is 100%, meaning that it always fails. This might result from a recent change introducing a regression that broke one of the realms. Another possibility, based on the realms provided by the platform, is that `wl_directUpdateRealm` is failing. This happens because of an issue with direct update. A third possibility is evident when an entry with [Unknown Realm] is displayed in the list of realms. This happens when there is a failure before one of the realms is reached. Having this information is especially useful when running an application in production.

Securing the Operational Analytics server

Learn about security with the IBM MobileFirst Platform Operational Analytics.

Protecting the analytics data entry point with basic authentication

IBM MobileFirst Platform Operational Analytics is configured to protect the data entry point by using basic authentication.

Default path protection

Data is sent from the MobileFirst Server to the IBM MobileFirst Platform Operational Analytics when the following IBM MobileFirst Platform Foundation for iOS property is set:

```
wl.analytics.url=http://<hostname>:<port>/analytics-service/data
```

The IBM MobileFirst Platform Operational Analytics exposes this path and it is protected by default using basic authentication (user name and password).

Configuration of basic authentication

In the `analytics-service` WAR file, basic authentication is configured in the `WEB-INF/web.xml` file. The default configuration has the following structure:

```
<!-- SECURITY ROLES -->
<security-role>
 <role-name>worklightadmin</role-name>
</security-role>
<security-role>
 <role-name>worklightdeployer</role-name>
</security-role>
<security-role>
 <role-name>worklightmonitor</role-name>
</security-role>
<security-role>
 <role-name>worklightoperator</role-name>
</security-role>

<!-- SECURITY CONSTRAINTS -->
<security-constraint>
 <web-resource-collection>
  <web-resource-name>allAccess</web-resource-name>
      <url-pattern>/data/*</url-pattern>
 </web-resource-collection>
 <auth-constraint>
  <role-name>worklightadmin</role-name>
  <role-name>worklightdeployer</role-name>
  <role-name>worklightmonitor</role-name>
  <role-name>worklightoperator</role-name>
 </auth-constraint>
  <user-data-contraint>
  <transport-guarantee>NONE</transport-guarantee>
 </user-data-constraint>
</security-constraint>

<!-- AUTHENTICATION METHOD: basic auth -->
<login-config>
 <auth-method>BASIC</auth-method>
 <realm-name>worklightRealm</realm-name>
</login-config>
```

Four roles are created (worklightadmin, worklightdeployer, worklightmonitor, worklightoperator) and the data endpoint is protected from all requests made by authenticated users not belonging to one of these roles.

Note: These role names also exist in the `analytics-ui` WAR file, which must also be edited if roles are added or changed.

Basic authentication credentials

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 11-61. After this configuration, the IBM MobileFirst Platform Operational Analytics will not accept any incoming data unless the request also contains the correct credentials. When the `wl.analytics.username` and `wl.analytics.password` values are set, the MobileFirst Server uses these credentials when it forwards data to the IBM MobileFirst Platform Operational Analytics. For more information about IBM MobileFirst Platform Foundation for iOS properties, see "MobileFirst properties" on page 13-90.

Note: This configuration protects only the `/data` path that accepts incoming data. It does not protect the console.

Ports that are used by the IBM MobileFirst Platform Operational Analytics

When the IBM MobileFirst Platform Operational Analytics is started, it listens on port 9600.

Port 9500 - HTTP Port

This port can be used for HTTP requests that are made directly to the IBM MobileFirst Platform Operational Analytics. 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 MobileFirst Platform Operational Analytics through this port. This port is closed by default. You can open this port by setting the `http.enabled` JNDI property to `true`. For more information, see "Properties and configurations" on page 13-90.

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 MobileFirst Platform Operational Analytics. It is not necessary to do any additional work to cluster the application server that the IBM MobileFirst Platform Operational Analytics is running on. The application servers are only necessary to host the IBM MobileFirst Platform Operational Analytics and do not require special configuration for clustering.

Creating an IBM MobileFirst Platform Operational Analytics 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 MobileFirst Platform Operational Analytics before you create the cluster.

Clustering terminology

Learn about clustering terminology for the IBM MobileFirst Platform Operational Analytics.

Cluster
A collection of one or more master and data nodes.

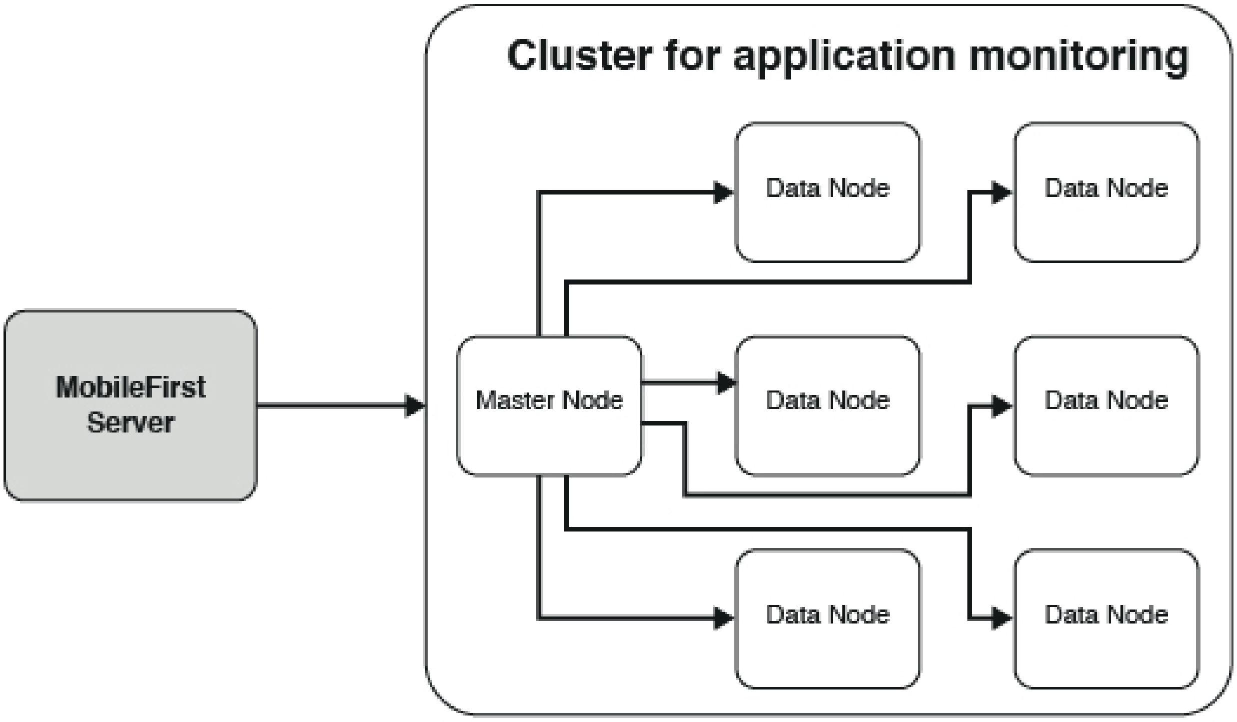
Master Node
Coordinator of the cluster. Manages the distribution of shards and keeps track of all nodes in the cluster. There can be more than one master node. If a master node fails, then a new node that is marked as a master node is automatically elected as a new master node. The cluster cannot operate 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 13-72.

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 13-78.

The following image shows a basic clustering topology:



Note: Data can be forwarded from the MobileFirst Server to any node in the server. The MobileFirst 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:

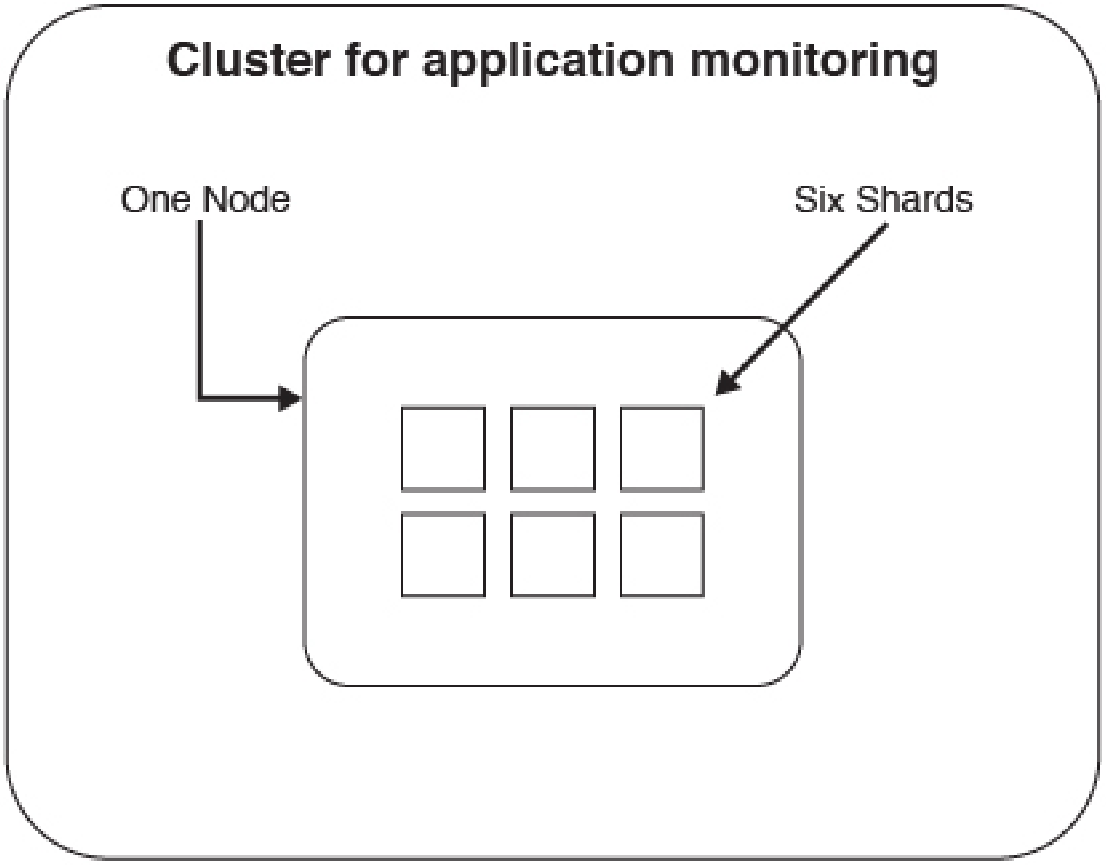
Learn about shards in the IBM MobileFirst Platform Operational Analytics.

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 MobileFirst Platform Operational Analytics.

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.

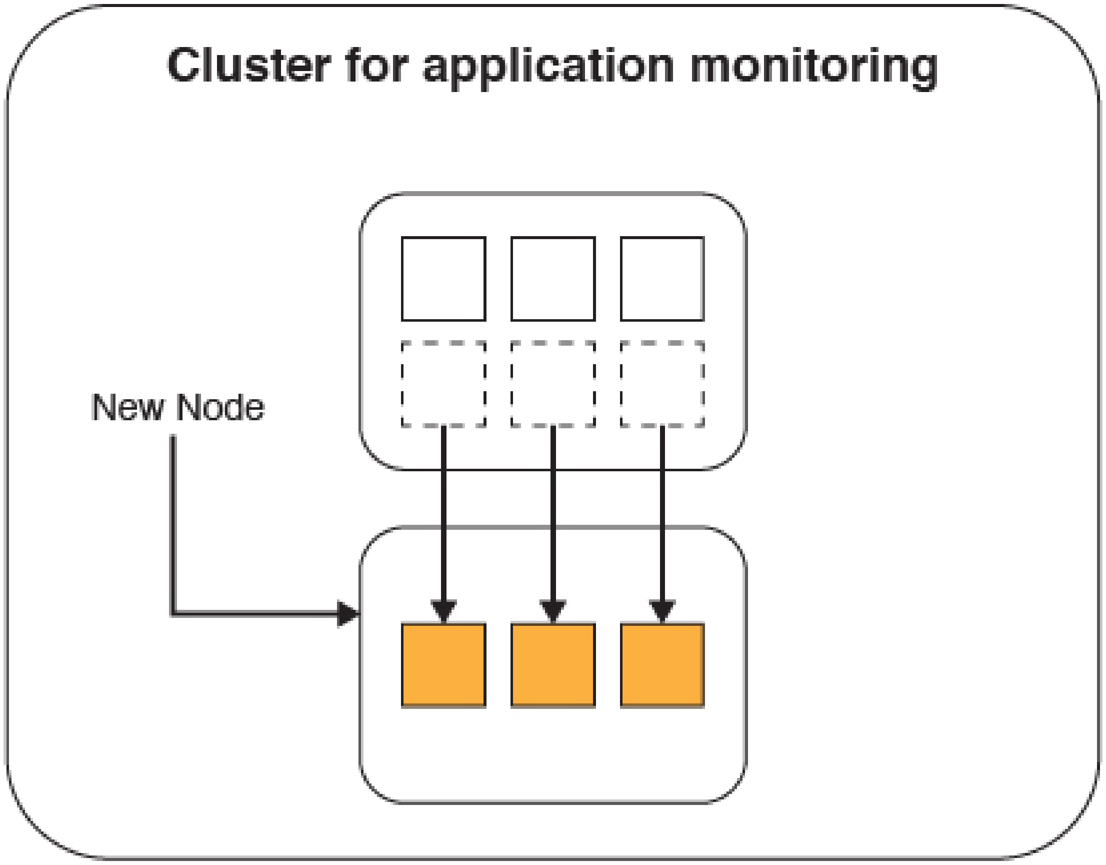
The following images show how sharding works.

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.

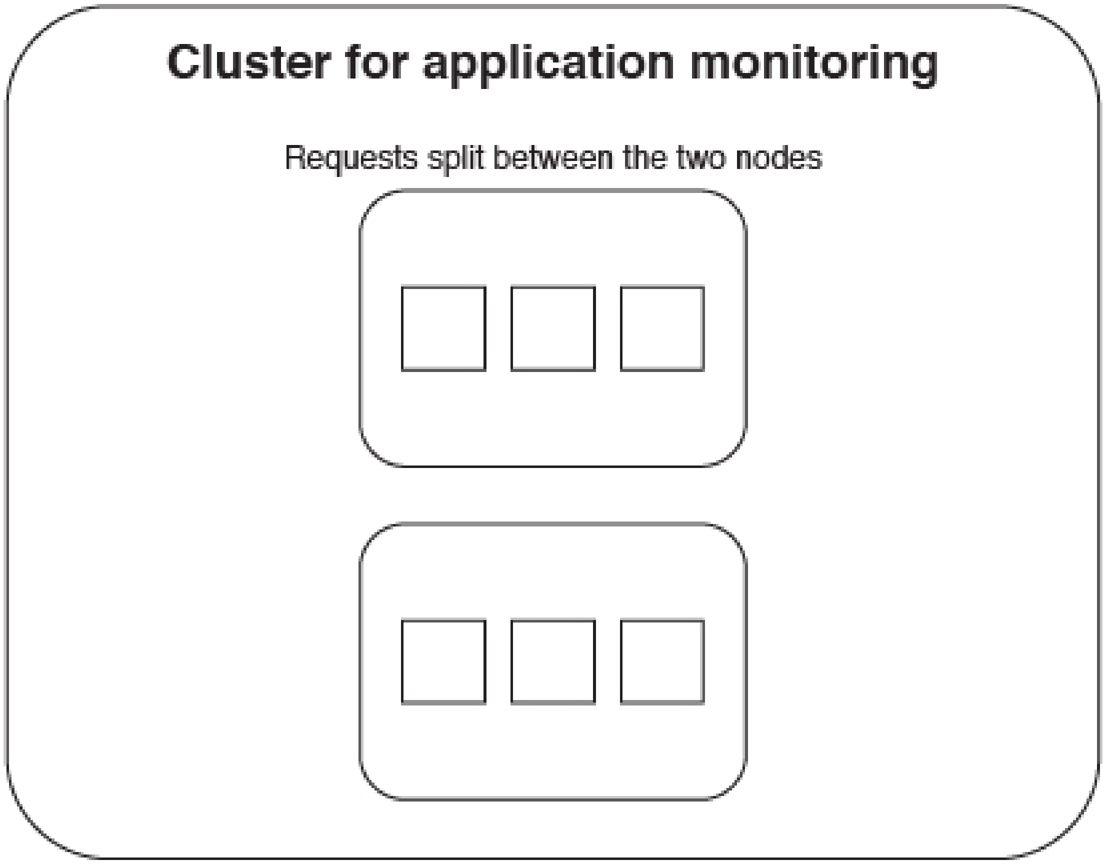


After several months of use, requests that are made to the cluster begin to perform poorly. It is determined that a single node is no longer adequate to handle processing for all of the incoming data. A new node is added to the cluster.

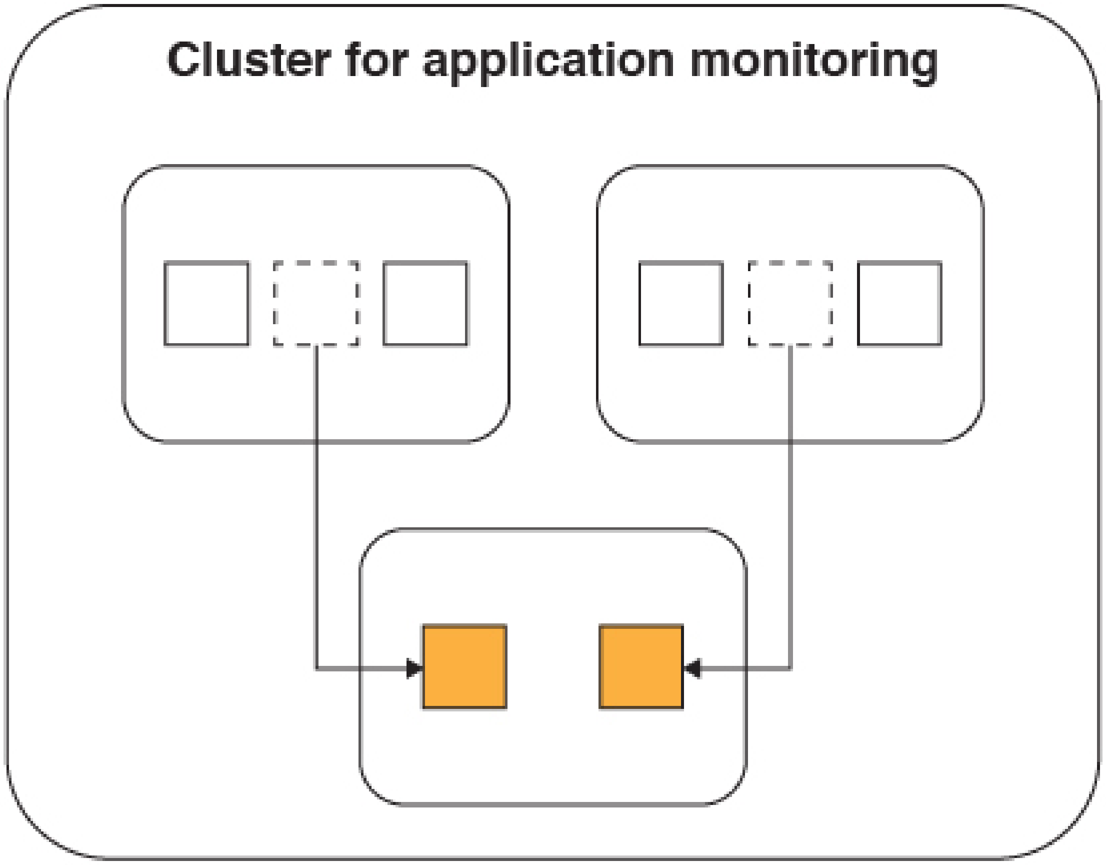
After a new node is added, the shards are automatically evenly distributed across all of the nodes.



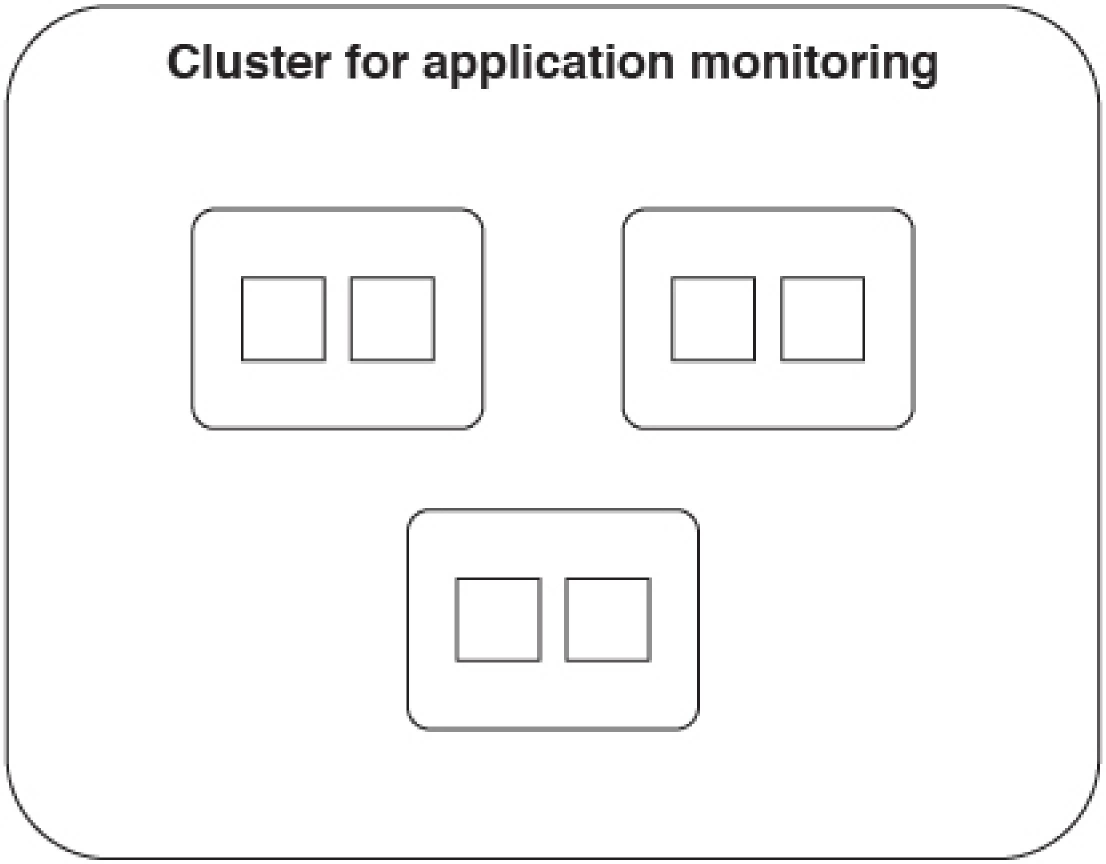
Now when a request comes in to either node, the request is forwarded to the node that has the shard that contains the data. The data indexing and processing is now split between the two nodes. Because the requests and data processing is now split between the nodes, the performance and response times improve.


Cluster for application monitoring
Requests split between the two nodes

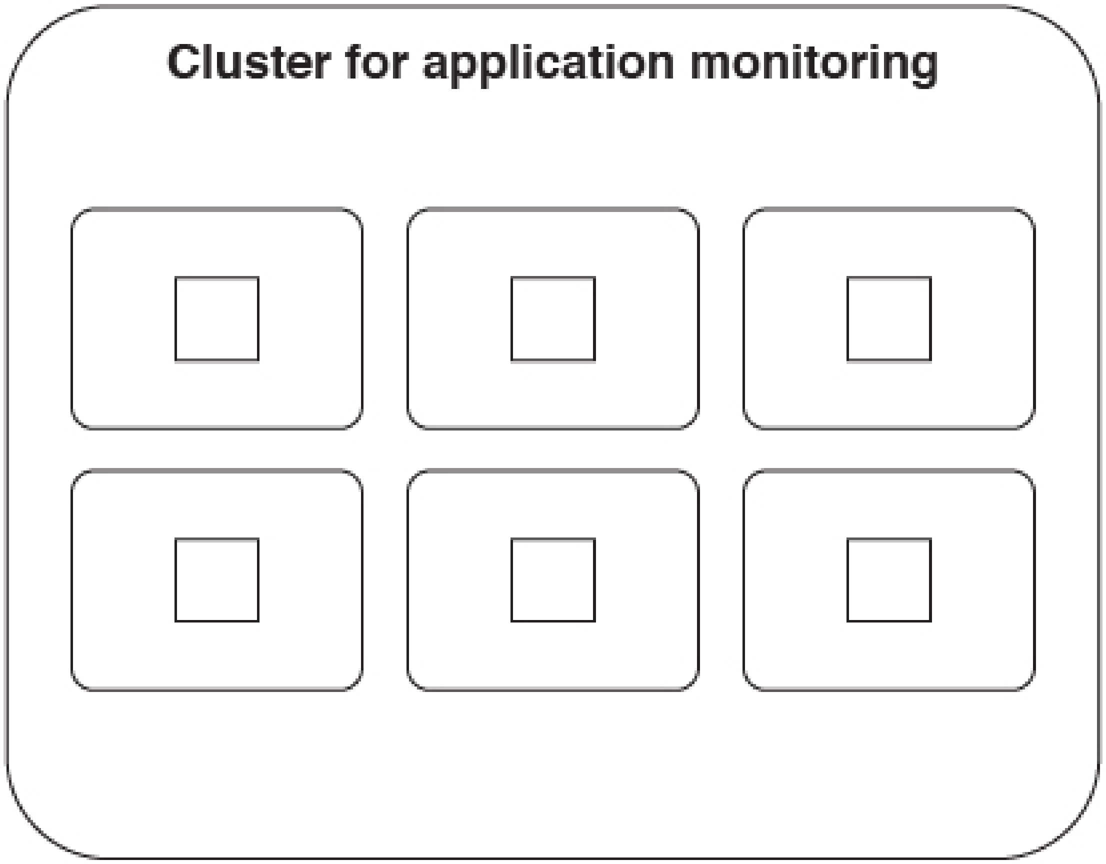
After several months, requests begin to slow down again, and it is determined that a third node is required.



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:

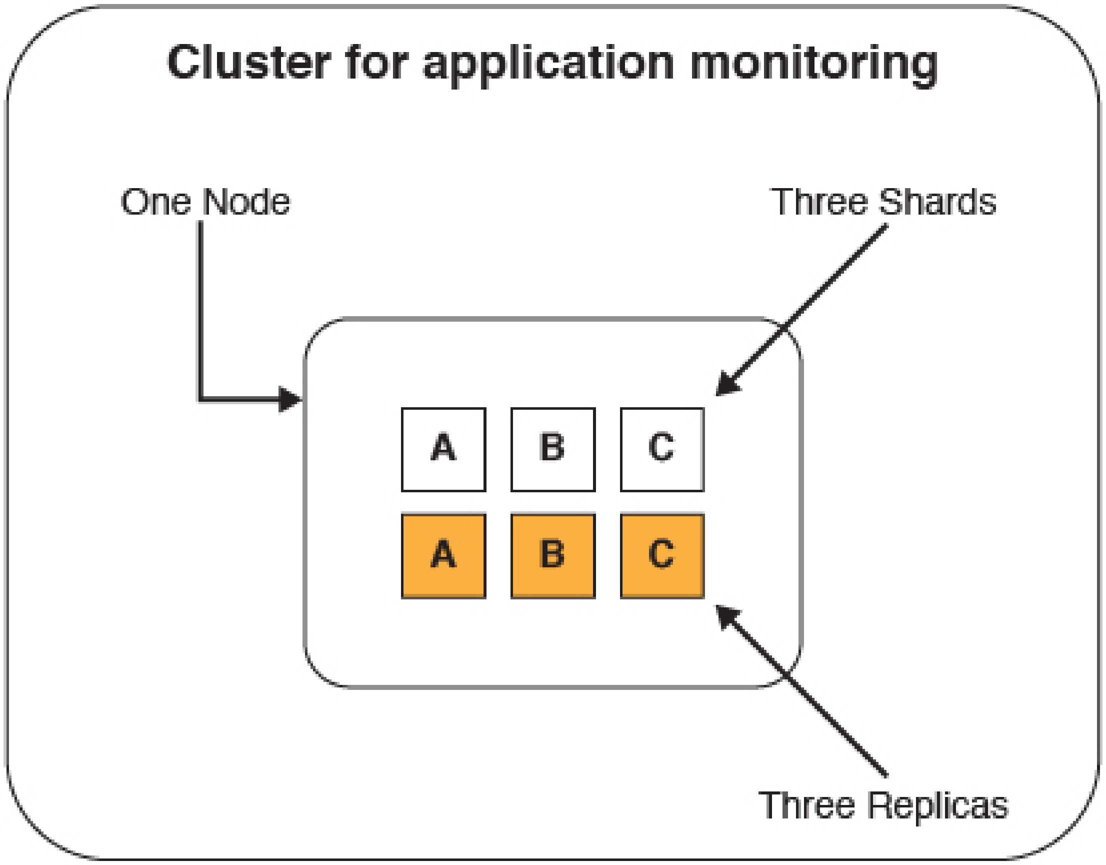
Learn about replicas in the IBM MobileFirst Platform Operational Analytics.

Shards contain the actual data that is sent from the MobileFirst 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 MobileFirst Platform Operational Analytics to create a specified number of replicas for each shard.

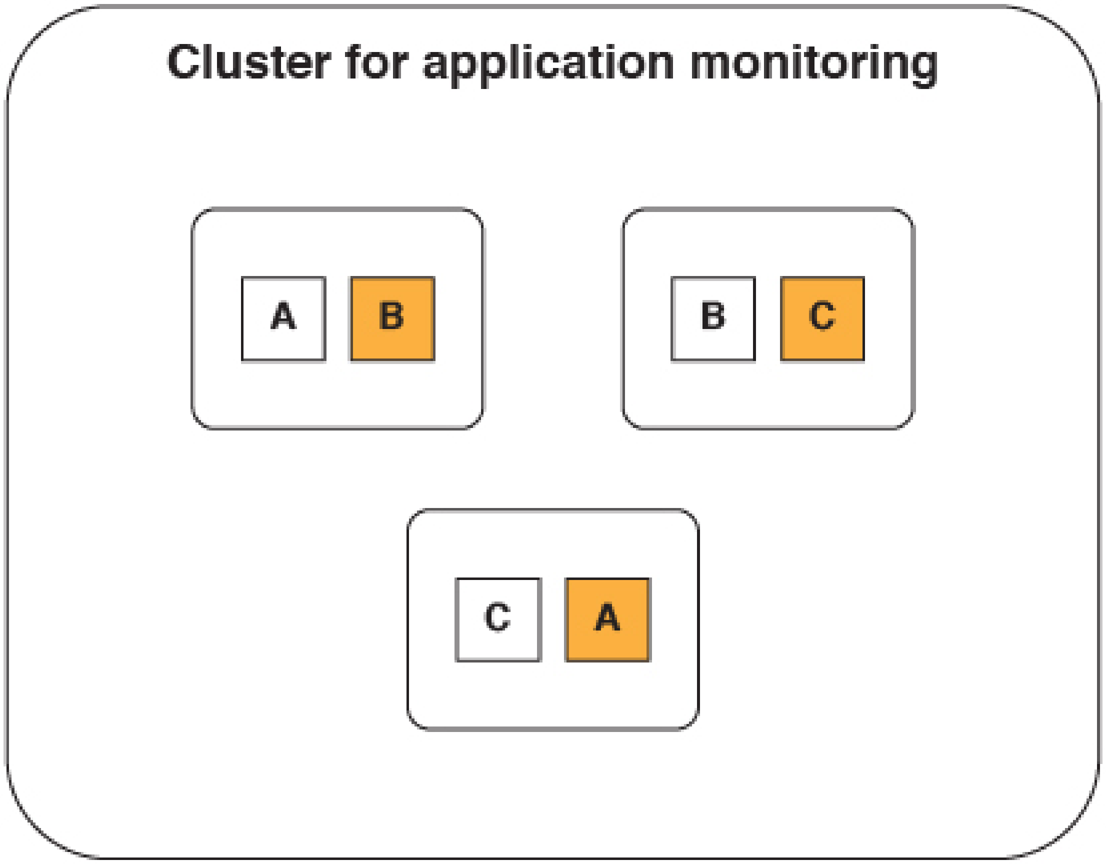
The following images show how replicas work.

Here is a cluster with one node, three shards, and one replica for each shard.

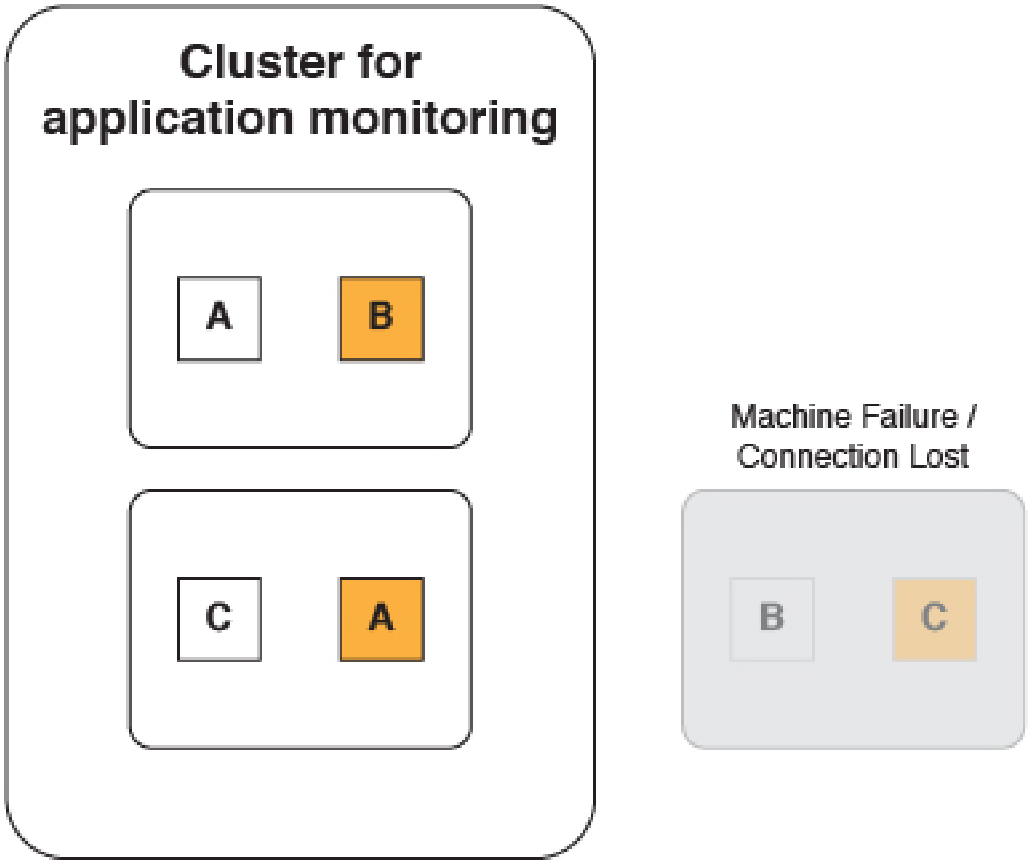


In this case, the replicas are redundant. Because there is only one node, having a replica exist on the same node does not accomplish anything. If the single node fails, the shards and replicas are all lost.

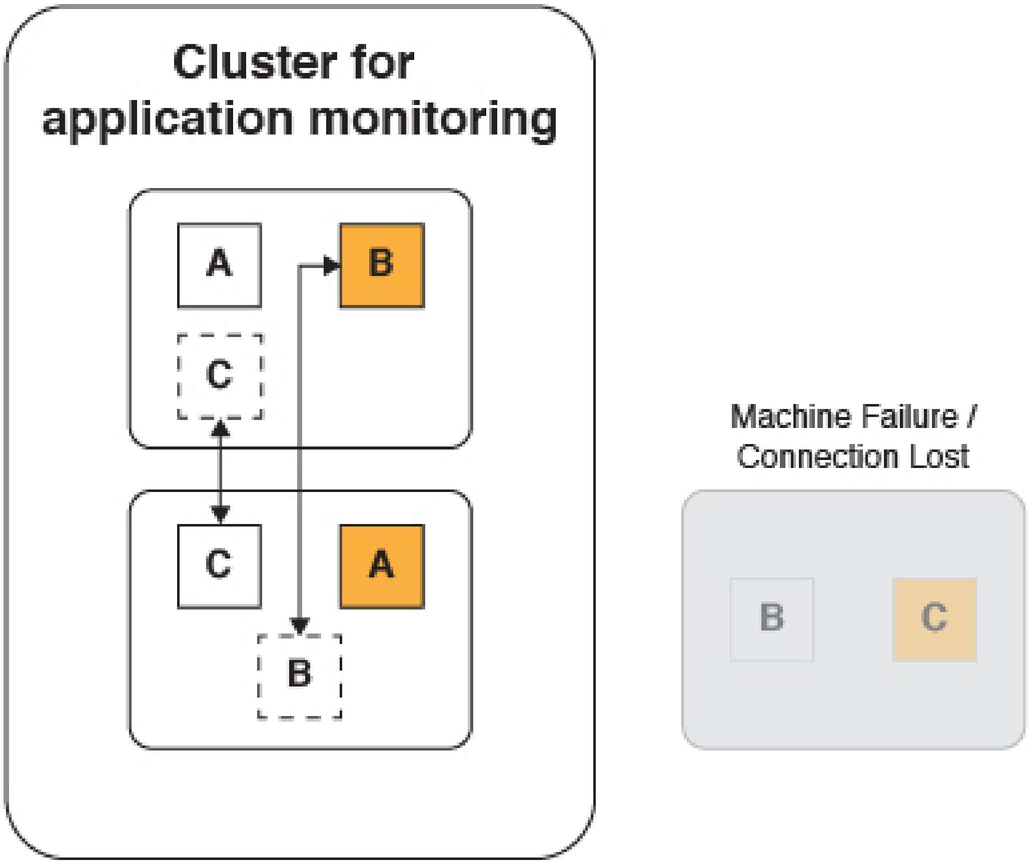
Now two more nodes are added to the cluster to improve performance:

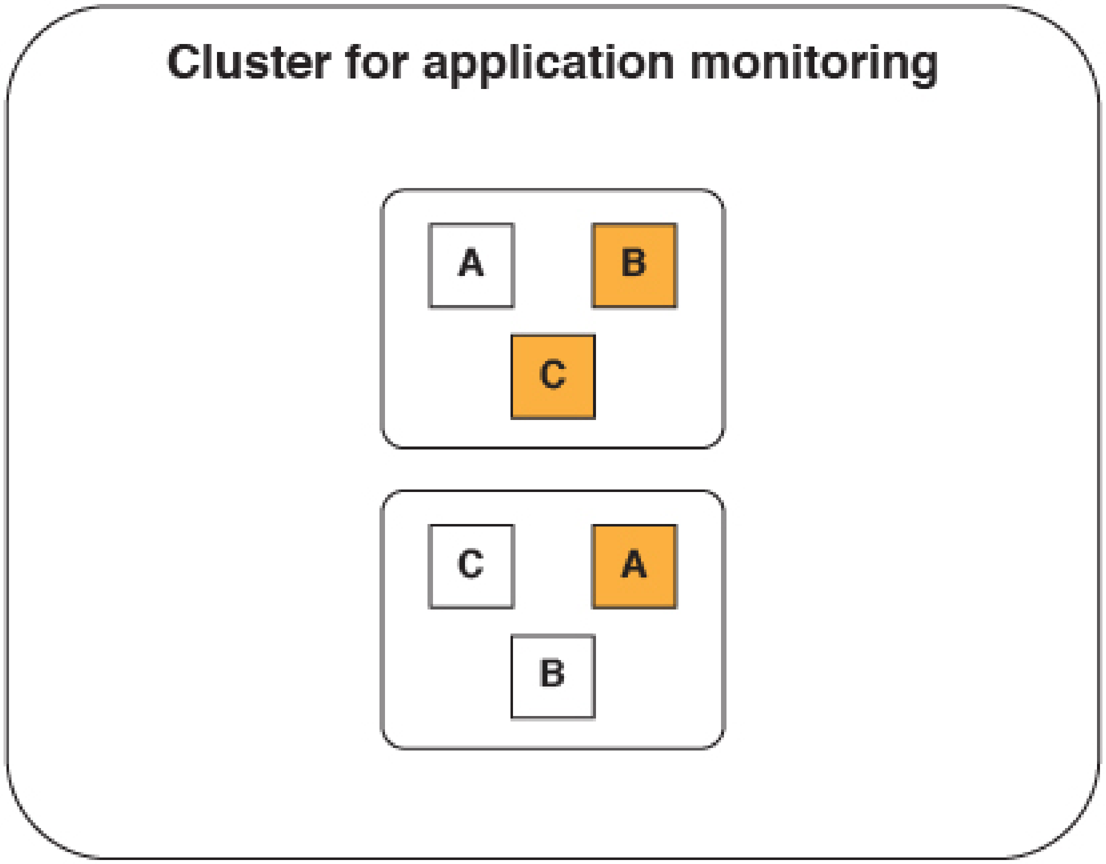


Notice that the shards and replicas are both automatically distributed evenly among the nodes. Now consider the scenario where a node fails due to a network or hardware issue.


Cluster for application monitoring
A
B
C
A
Machine Failure / Connection Lost
B
C

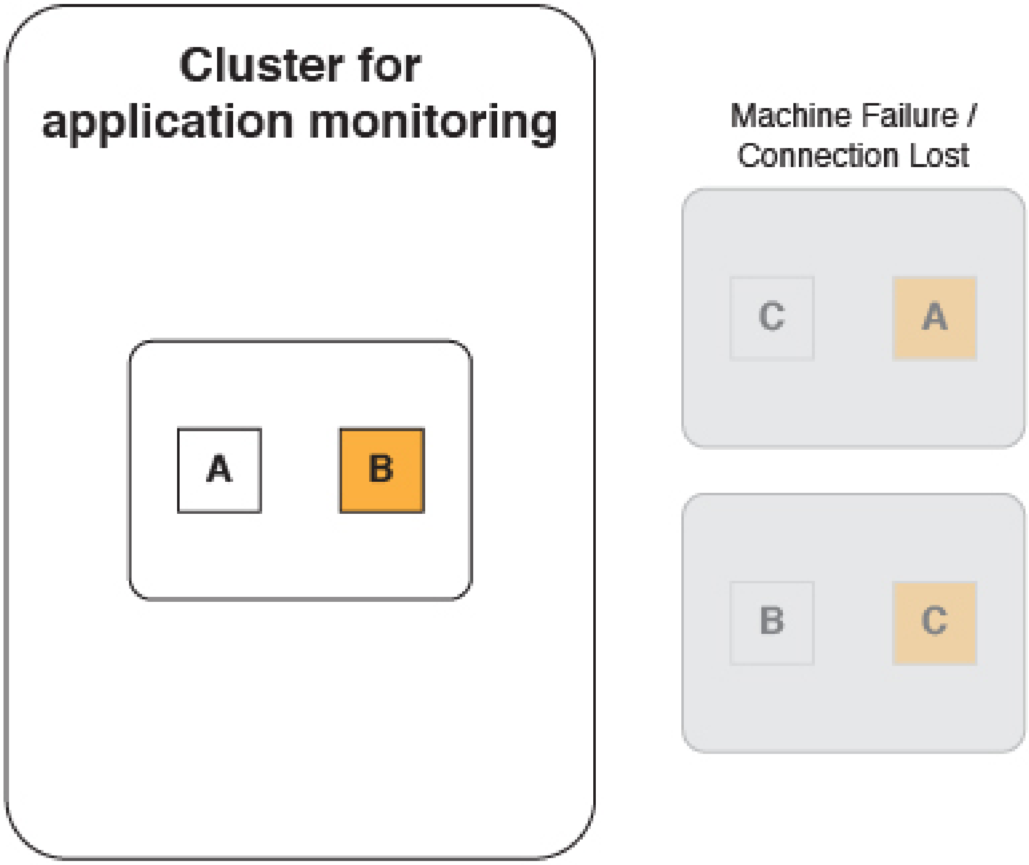
The analytics cluster can still operate normally because a replica of the lost shards still exists on one of the remaining nodes.


Cluster for application monitoring
A
B
C
C
A
B
Machine Failure / Connection Lost
B
C



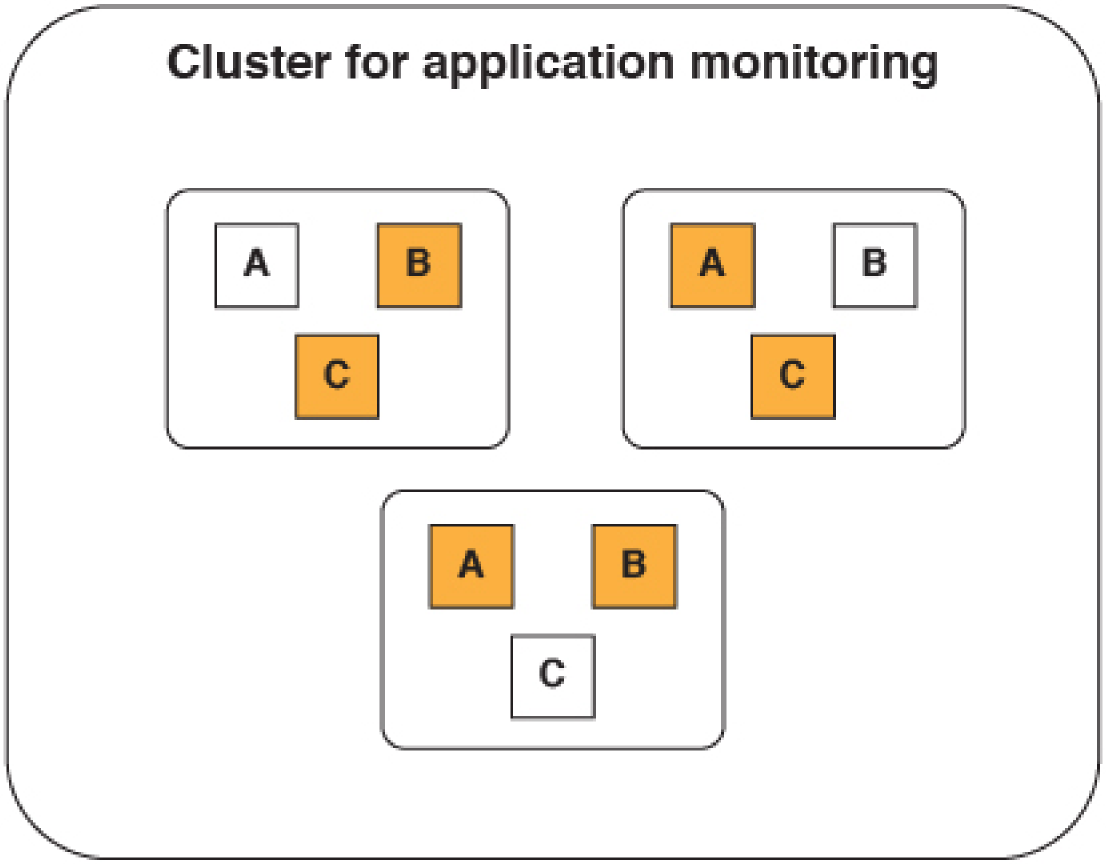
When the failed node comes back online and rejoins the cluster, the shards and replicas are again evenly distributed. The cluster returns to the state it was in before one of the nodes failed.

But what happens when two of the nodes fail simultaneously?


Cluster for application monitoring
A
B
Machine Failure / Connection Lost
C
A
B
C

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 MobileFirst Platform Operational Analytics on the same server as the MobileFirst Server. The IBM MobileFirst Platform Operational Analytics 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 13-87.

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 MobileFirst Platform Operational Analytics. 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 MobileFirst Platform Operational Analytics, 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 MobileFirst Platform Operational Analytics.
   - Set the minimum and maximum heap size to the same value for an application server that is hosting the IBM MobileFirst Platform Operational Analytics.
   - To change the heap size on WebSphere Application Server, from the console, go to WebSphere Application Servers/<*server name*>/Java, and Process Management/Process definition/Java Virtual Machine. Then, restart the server for changes to be taken into account.

   For an IBM MobileFirst Platform Operational Analytics 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 MobileFirst Platform Operational Analytics" on page 13-70.

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>/analytics-service/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.

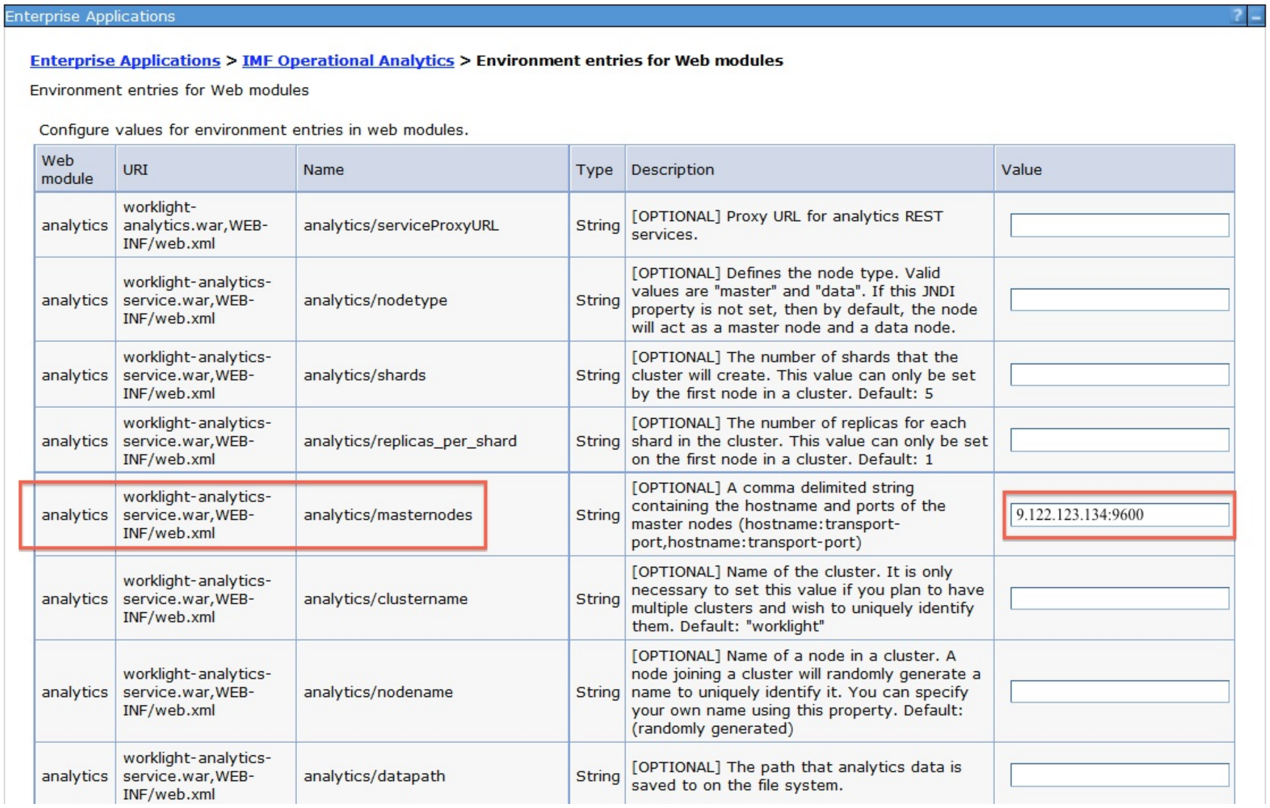
To use the Analytics Platform in a clustered environment on WebSphere Application Server, follow these steps.

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 MobileFirst Platform Operational Analytics" on page 13-70.
   The following image shows the environment entries for web modules:



Enterprise Applications > IMF Operational Analytics > 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	worklight-analytics.war,WEB-INF/web.xml	analytics/serviceProxyURL	String	[OPTIONAL] Proxy URL for analytics REST services.	
analytics	worklight-analytics-service.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	worklight-analytics-service.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	worklight-analytics-service.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	worklight-analytics-service.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
analytics	worklight-analytics-service.war,WEB-INF/web.xml	analytics/clustername	String	[OPTIONAL] Name of the cluster. It is only necessary to set this value if you plan to have multiple clusters and wish to uniquely identify them. Default: "worklight"	
analytics	worklight-analytics-service.war,WEB-INF/web.xml	analytics/nodename	String	[OPTIONAL] Name of a node in a cluster. A node joining a cluster will randomly generate a name to uniquely identify it. You can specify your own name using this property. Default: (randomly generated)	
analytics	worklight-analytics-service.war,WEB-INF/web.xml	analytics/datapath	String	[OPTIONAL] The path that analytics data is saved to on the file system.	

4. Set the remaining JNDI properties. For more information about the JNDI properties, see “Properties and configurations” on page 13-90.
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 MobileFirst Platform Operational Analytics.

Java virtual machine (JVM) swapping

The underlying technology that is used by the IBM MobileFirst Platform Operational Analytics 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" />
```

Setting the Field Cache Size

The underlying technology used by the analytics platform loads several field values into memory to provide fast access to those documents. This is known as the field cache. By default, the amount of data loaded into memory by the field cache is unbounded. If the field cache becomes too large, it can cause an out of memory exception and crash the analytics platform. You can put an upper limit on the field cache to prevent this from happening. The field cache can be set using the following JNDI property:

```
<jndiEntry jndiName="analytics/indices.fielddata.cache.size" value="80%"/>
<jndiEntry jndiName="analytics/indices.fielddata.cache.size" value="10GB"/>
```

Note: The field cache can be set using a hardcoded value (such as 10GB) or it can be set to a percentage of the heap (such as 80%).

Placing an upper limit on the cache will prevent the field cache from causing an out of memory exception. However, when the limit is reached, the analytics platform will begin to take longer to perform search queries. At this point, you can either add additional memory to your machine or add an additional node to your cluster.

Properties and configurations

Learn about the properties and configurations that are used for configuring the MobileFirst Server and IBM MobileFirst Platform Operational Analytics.

MobileFirst properties

These properties can be set on the MobileFirst 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 “JNDI environment entries for MobileFirst projects in production” on page 11-67.

For more information about the IBM MobileFirst Platform Operational Analytics properties, see “Analytics” on page 11-58.

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 13-85. 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 13-21. JNDI properties for the IBM MobileFirst Platform Operational Analytics. This table lists the JNDI property names, default values, and descriptions for the IBM MobileFirst Platform Operational Analytics.

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 per index 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 host name and ports of the master nodes. For more information about this property, see "Setting up a production cluster" on page 13-85.
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.

Table 13-21. JNDI properties for the IBM MobileFirst Platform Operational Analytics (continued). This table lists the JNDI property names, default values, and descriptions for the IBM MobileFirst Platform Operational Analytics.

Property Name	Default Value	Description
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 13-93.
transportport	9600	Port that is used for node-to-node communication. For more information, see "Ports that are used by the IBM MobileFirst Platform Operational Analytics" on page 13-70.
httpport	9500	Port that is used for HTTP communication to the IBM MobileFirst Platform Operational Analytics. For more information, see "Ports that are used by the IBM MobileFirst Platform Operational Analytics" on page 13-70.
http.enabled	false	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.

Table 13-21. JNDI properties for the IBM MobileFirst Platform Operational Analytics (continued). This table lists the JNDI property names, default values, and descriptions for the IBM MobileFirst Platform Operational Analytics.

Property Name	Default Value	Description
serviceProxyURL	None	This property enables the IBM MobileFirst Platform Operational Analytics console to locate the Analytics REST services. The value of this property must be specified as the external address and context root of the worklight-analytics-service.war web application.
bootstrap.mlockall	true	This property prevents any Elasticsearch memory from being swapped out.
index.mapper.dynamic	false	This property allows for the dynamic creation of mappings for unmapped types to be disabled.
multicast	false	Enables or disables multicast ping discovery.
warmupFrequencyInSeconds	600	This property runs background queries immediately on start in the specified interval to force data query results into memory, improving web console performance. Negative value disables running the background warmup.
tenant	worklight	Name of the main ElasticSearch index.

Elasticsearch properties

Elasticsearch is the underlying technology that is used by IBM MobileFirst Platform Operational Analytics. 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" />
```

Analytics Console properties

For more information about the settings that can be set in the **Update Settings** page of the Analytics Console, including TTL settings, see "Update Settings" on page 13-18 and "Data purging" on page 13-20.

Backing up Operational Analytics data

Learn about how to back up your MobileFirst Operational Analytics data.

The data for MobileFirst Operational Analytics is stored as a set of files on the MobileFirst Operational Analytics Server file system. The location of this folder is specified by the `datapath` JNDI property in the MobileFirst Operational Analytics Server configuration. For more information about the JNDI properties, see "Properties and configurations" on page 13-90.

The MobileFirst Operational Analytics Server configuration is also stored on the file system, and is called `server.xml`.

You can back up these files by using any existing server backup procedures that you might already have in place. No special procedure must be used when you back up these files, other than ensuring that the MobileFirst Operational Analytics Server is stopped. Otherwise, the data might change while the backup is occurring, and the data that is stored in memory might not yet be written to the file system. To avoid inconsistent data, stop the MobileFirst Operational Analytics Server before you start your backup.

(deprecated) Reports database

IBM MobileFirst Platform Foundation for iOS provides an extensible mechanism for enterprises to use to integrate reporting tools with IBM MobileFirst Platform Foundation for iOS.

Note: The Reports database and the sample BIRT Reports were deprecated in IBM MobileFirst Platform Foundation for iOS V7.0.0. Use "Operational analytics" on page 13-6 instead.

IBM MobileFirst Platform Foundation for iOS provides raw data reports and a number of device reports that are aggregated from the raw data report table. IBM MobileFirst Platform Foundation for iOS 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 13-97.

Note: Enabling the BIRT feature is redundant if you already use the IBM MobileFirst Platform Operational Analytics.

IBM MobileFirst Platform Foundation for iOS provides three reporting mechanisms:

Raw data feeds
IBM MobileFirst Platform Foundation for iOS 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 13-98.

Device usage reports

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS with an OLAP system. For more information, see “Device usage reports” on page 13-102.

Note: Device usage reports are functional only in IBM MobileFirst Platform Foundation for iOS Customer Edition and IBM MobileFirst Platform Foundation for iOS Enterprise Edition.

BIRT reports

IBM MobileFirst Platform Foundation for iOS 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 13-105.

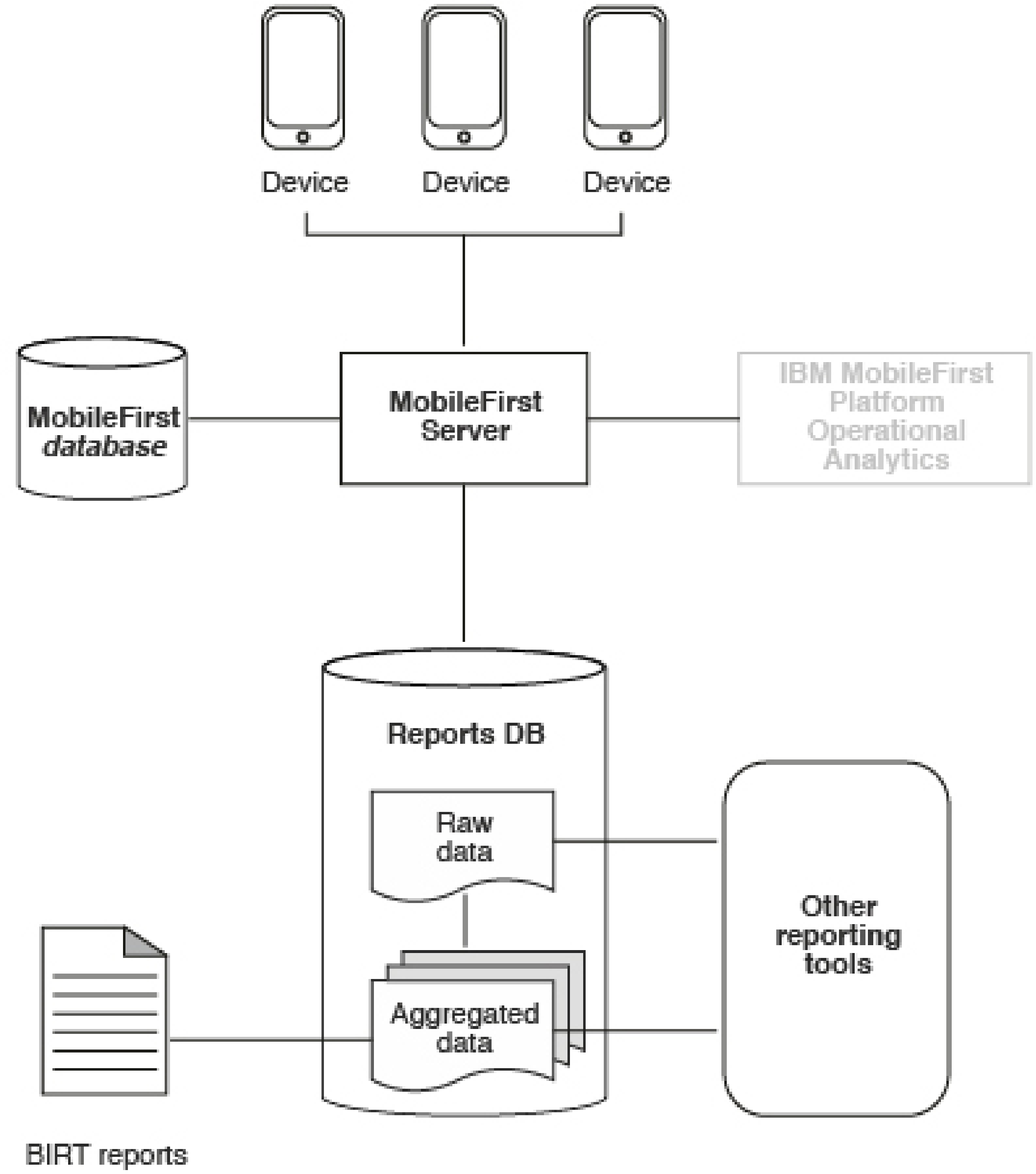


Figure 13-27. High-level overview of the reports architecture

The reports architecture diagram shows how the raw data feed comes from three devices into the MobileFirst Server and then into the IBM MobileFirst Platform Foundation for iOS 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.

Comparison of operational analytics and reports features

Compare the reports and operational analytics features to know why and how to best use each.

Note: The Reports database and the sample BIRT Reports were deprecated in IBM MobileFirst Platform Foundation for iOS V7.0.0. Use "Operational analytics" on page 13-6 instead.

With the introduction of the IBM MobileFirst Platform Operational Analytics, 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 "(deprecated) Reports database" on page 13-94 feature is a subset of the total data that is collected as part of the "Operational analytics" on page 13-6 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 13-22. 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, MobileFirst Server log, MobileFirst app to server interaction activities	MobileFirst app to server interaction activities
Data storage mechanism	Files on the IBM MobileFirst Platform Operational Analytics	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), time stamp, 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 time stamp relationships.

Table 13-22. Comparison of analytics and reports features (continued). This table lists a comparison of analytics and reports features.

	Operational analytics feature	Reports feature
Access mechanism	MobileFirst Operations 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 MobileFirst Platform Operational Analytics.

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

Note: The Reports database and the sample BIRT Reports were deprecated in IBM MobileFirst Platform Foundation for iOS V7.0.0. Use "Operational analytics" on page 13-6 instead.

Raw data reports provide you with analytics information about your applications and adapter usage, such as activity type, device information, and application version. Use the following steps to enable the raw data reports feature:

Procedure

1. Ensure that the IBM MobileFirst Platform 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 "Runtime database setup for development mode" on page 11-52. For information about creating databases and schemas in a production environment, see "Creating and configuring the databases manually" on page 11-20.
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.

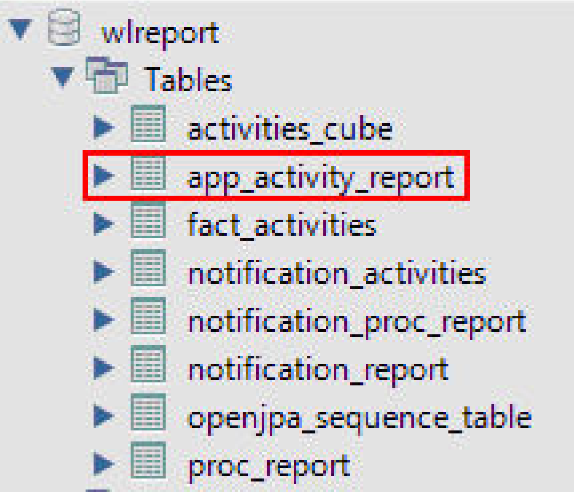
```
#################################################
# Raw reports
#################################################
reports.exportRawData=true
# jndi name; empty value means Apache DBCP data source
#wl.reports.db.jndi.name=${wl.db.jndi.name}
# Default values for DBCP connection pool
#wl.reports.db.initialSize=${wl.db.initialSize}
#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 MobileFirst project in production by using JNDI environment entries" on page 11-65.

5. Restart your application server.

   The app_activity_report table of the raw data database is populated with data as you use your applications and adapters.



wlreport
Tables
activities_cube
app_activity_report
fact_activities
notification_activities
notification_proc_report
notification_report
openjpa_sequence_table
proc_report

The raw data app_activity_report table contains the following information:

Column	Description
ACTIVITY_TIMESTAMP	UTC time of entry
GADGET_NAME	MobileFirst Application name
GADGET_VERSION	Application version
ACTIVITY	Activity type
ENVIRONMENT	Application environment name (iPhone, and so on)
SOURCE	User identifier
ADAPTER	MobileFirst adapter name
PROC	MobileFirst adapter procedure name

Column	Description
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 iPhone 5
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
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: MobileFirst 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):

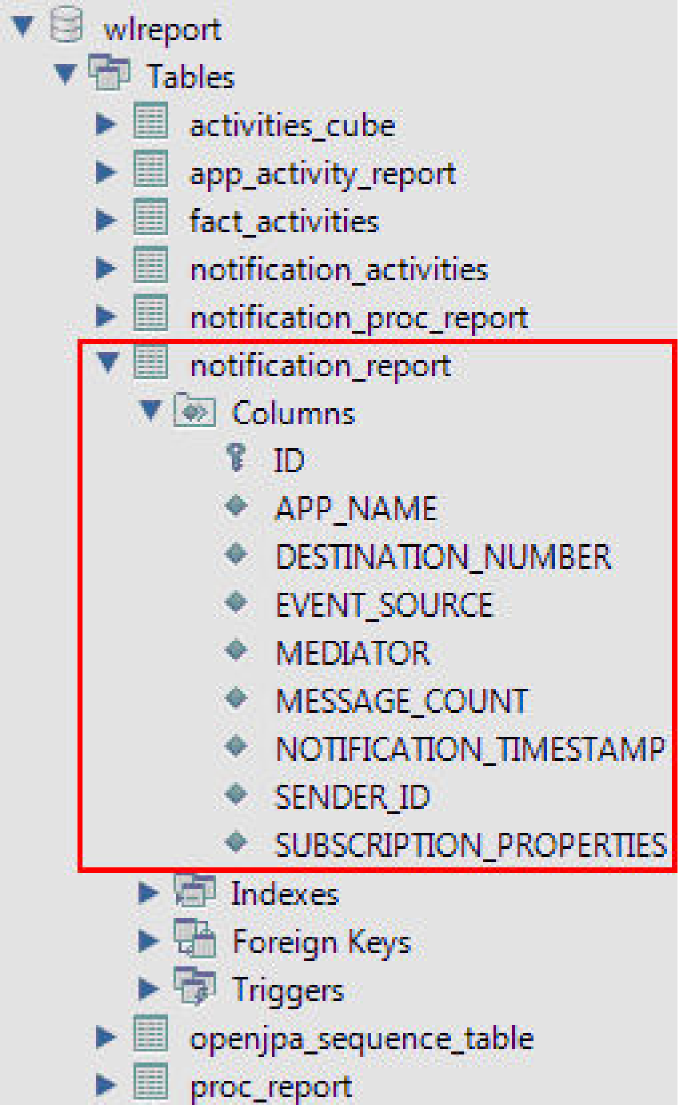
```
DELETE FROM ACTIVITIES_CUBE WHERE ACTIVITY_DATE <= TRUNC(SYSDATE) - 30
```

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) - 30)
```

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 MobileFirst Server project databases" on page 6-142.

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.



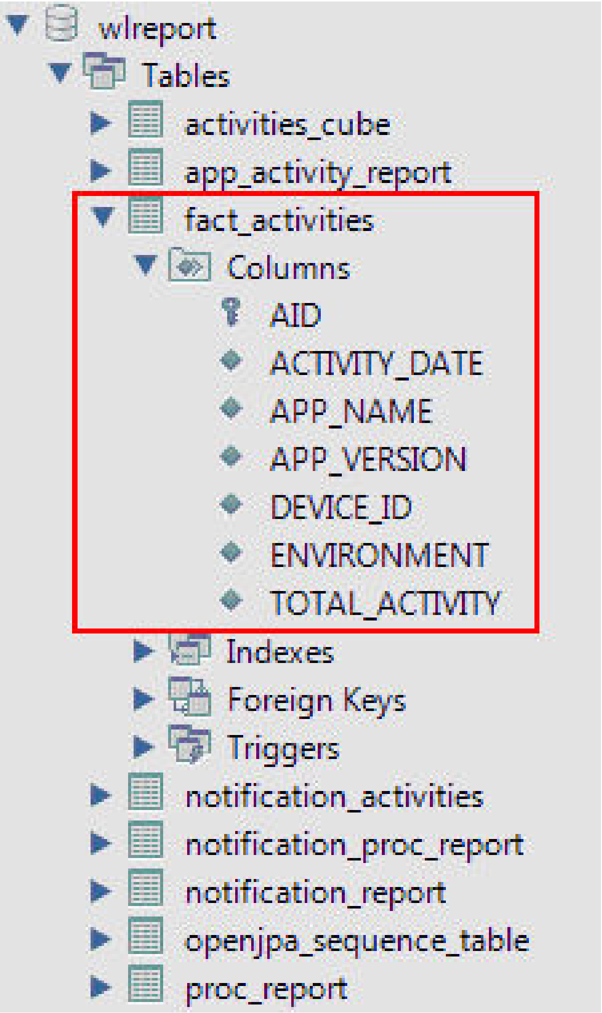
Device usage reports

For simpler and faster access to the reports data, IBM MobileFirst Platform Server runs an analytics data processor task at a default time interval of every 24 hours.

Note: The Reports database and the associated `APP_ACTIVITY_REPORT` table described below are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use “Operational analytics” on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

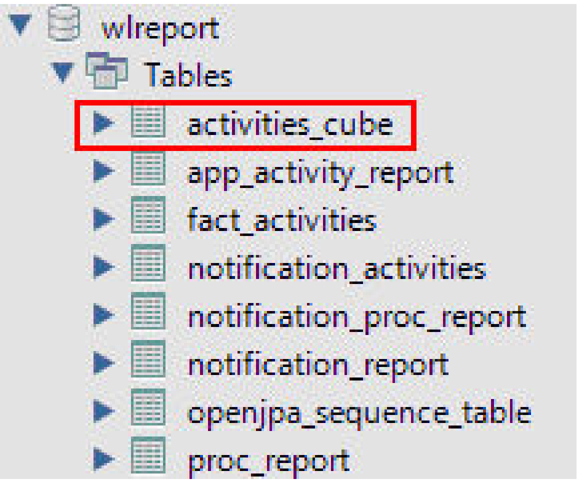
The analytics data processor task retrieves raw entries for the specified time interval from the `app_activity_report` table and processes them to populate the `fact_activities` table.

Note: The `fact_activities` table is only populated with usage data from hybrid and native applications from actual devices. Usage data from MobileFirst mobile web applications that are running on actual devices or from a browser, such as when you are using preview, is not populated into this table.

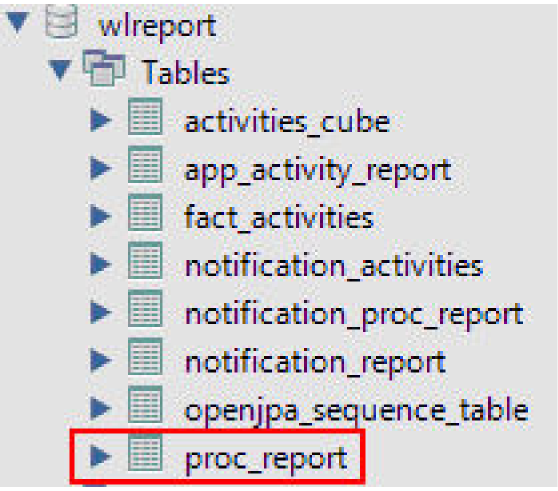


- wlreport
  - Tables
    - activities_cube
    - app_activity_report
    - fact_activities
      - Columns
        - AID
        - ACTIVITY_DATE
        - APP_NAME
        - APP_VERSION
        - DEVICE_ID
        - ENVIRONMENT
        - TOTAL_ACTIVITY
      - Indexes
      - Foreign Keys
      - Triggers
    - notification_activities
    - notification_proc_report
    - notification_report
    - openjpa_sequence_table
    - proc_report

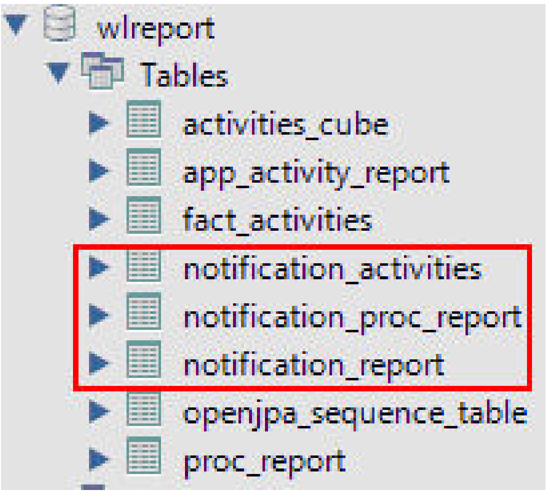
The `fact_activities` table contains a total activity count (number of logged actions) per application, application version, device, and environment. The `fact_activities` data is also processed and put into the `activities_cube` table. This table has the same structure as the `fact_activities` table and only contains records for the last 30 days.



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.



wlreport
Tables
activities_cube
app_activity_report
fact_activities
notification_activities
notification_proc_report
notification_report
openjpa_sequence_table
proc_report

The processing interval can be modified by adding the following property to your `worklight.properties` file and setting the required interval in seconds.

```
# Default interval value for analytics processing task
wl.db.factProcessingInterval=86400
```

The processing interval can also be disabled by setting this property to a negative value.

```
# Set to a negative value to disable the analytics processing task
wl.db.factProcessingInterval=-1
```

Predefined BIRT Reports

You can use predefined BIRT reports to generate and display information about mobile devices and usage.

Note: The predefined BIRT reports and the Reports database and associated tables such as `APP_ACTIVITY_REPORT` are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

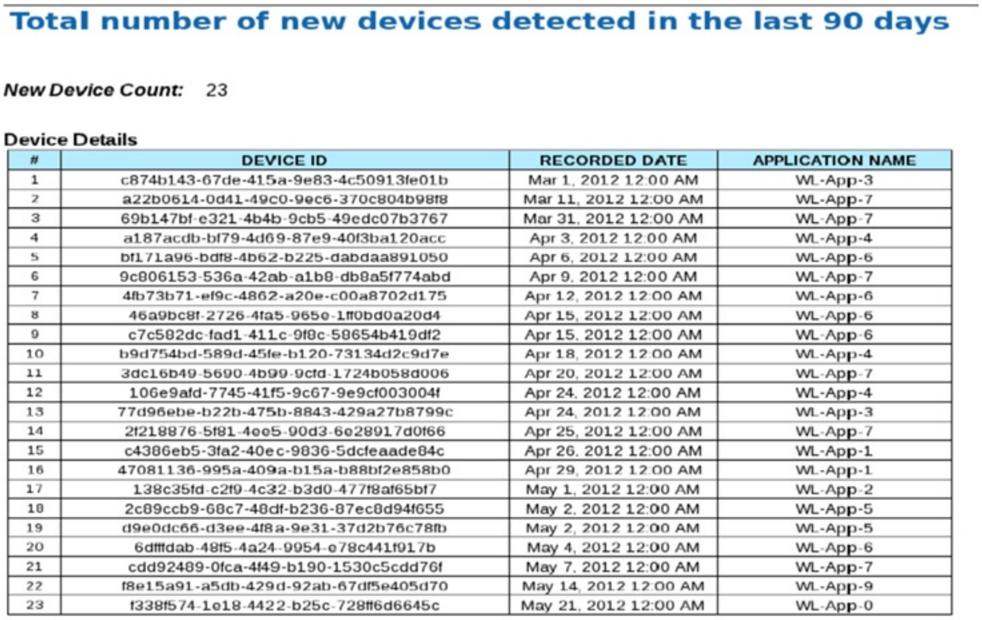
IBM MobileFirst Platform Foundation for iOS generates raw reports, which are stored in an `app_activity_report` table. IBM MobileFirst Platform Foundation for iOS also includes device usage reports, which are aggregations of data from the `app_activity_report`, and are described in "Device usage reports" on page 13-102 and "Using raw data reports" on page 13-98. 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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS provides several `*.rptdesign` files that contain logic to connect to the reports database, pull data from device usage tables, process, and display the data.

The IBM MobileFirst Platform Foundation editions containing MobileFirst Server include the following predefined BIRT reports:

Table 13-23. Predefined BIRT reports

Report Name	Description	Report file name
Active Users	Active users in last 30 days.	`report_active_users.rptdesign`
Daily Hits	The daily aggregated hits for last 30 days. Any action from the user/device that caused a request to the server is counted as a hit. This number, aggregated over a day, equals the daily hits.	`report_daily_hits.rptdesign`
Daily Visits	The number of discreet visits by separate user/device in last 30 days. All actions by a user/device that caused one or more requests to the server within a day is counted as a visit.	`report_daily_visits.rptdesign`
Environment Usage	Application version and application environment used: number of visits that were recorded in the last 30 days.	`report_environment_usage.rptdesign`
New Devices	A record of unique devices that were connected in the last 30 days.	`report_new_devices.rptdesign`
Notification Messages Per Day	Number of messages sent each day in the past 90 days per data source.	`report_notification_messages_per_day.rptdesign`
Notification Messages Per Source	Total number of messages that were sent in the last 90 days per data source.	`report_notification_messages_per_source.rptdesign`
License Total New Device Count	A record of unique devices that were connected over a specified period (90 days as default), for licensing purposes.	`report_license_total_device_count.rptdesign`



Total number of new devices detected in the last 90 days

New Device Count: 23

Device Details

#	DEVICE ID	RECORDED DATE	APPLICATION NAME
1	c874b143-67de-415a-9e83-4c50913fe01b	Mar 1, 2012 12:00 AM	WL-App-3
2	a22b0614-0d41-49c0-9ec6-370c804b98f8	Mar 11, 2012 12:00 AM	WL-App-7
3	69b147bf-e321-4b4b-9cb5-49edc07b3767	Mar 31, 2012 12:00 AM	WL-App-7
4	a187acdb-bf79-4d69-87e9-40f3ba120acc	Apr 3, 2012 12:00 AM	WL-App-4
5	bf171a96-bdf8-4b62-b225-dabdaa891050	Apr 6, 2012 12:00 AM	WL-App-6
6	9c806153-536a-42ab-a1b8-db8a5f774abd	Apr 9, 2012 12:00 AM	WL-App-7
7	4fb73b71-ef9c-4862-a20e-c00a8702d175	Apr 12, 2012 12:00 AM	WL-App-6
8	46a9bc8f-2726-4fa5-965e-1ff0bd0a20d4	Apr 15, 2012 12:00 AM	WL-App-6
9	c7c582dc-fad1-411c-9f8c-58654b419df2	Apr 15, 2012 12:00 AM	WL-App-6
10	b9d754bd-589d-45fe-b120-73134d2c9d7e	Apr 18, 2012 12:00 AM	WL-App-4
11	3dc16b49-5690-4b99-9cfd-1724b058d006	Apr 20, 2012 12:00 AM	WL-App-7
12	106e9afd-7745-41f5-9c67-9e9cf003004f	Apr 24, 2012 12:00 AM	WL-App-4
13	77d96ebe-b22b-475b-8843-429a27b8799c	Apr 24, 2012 12:00 AM	WL-App-3
14	2f218876-5f81-4ee5-90d3-6e28917d0f66	Apr 25, 2012 12:00 AM	WL-App-7
15	c4386eb5-3fa2-40ec-9836-5dcfeaade84c	Apr 26, 2012 12:00 AM	WL-App-1
16	47081136-995a-409a-b15a-b88bf2e858b0	Apr 29, 2012 12:00 AM	WL-App-1
17	138c35fd-c2f9-4c32-b3d0-477f8af65bf7	May 1, 2012 12:00 AM	WL-App-2
18	2c89ccb9-68c7-48df-b236-87ec8d94f655	May 2, 2012 12:00 AM	WL-App-5
19	d9e0dc66-d3ee-4f8a-9e31-37d2b76c78fb	May 2, 2012 12:00 AM	WL-App-5
20	6dfffdab-48f5-4a24-9954-e78c441f917b	May 4, 2012 12:00 AM	WL-App-6
21	cdd92489-0fca-4f49-b190-1530c5cdd76f	May 7, 2012 12:00 AM	WL-App-7
22	f8e15a91-a5db-429d-92ab-67df5e405d70	May 14, 2012 12:00 AM	WL-App-9
23	f338f574-1e18-4422-b25c-728ff6d6645c	May 21, 2012 12:00 AM	WL-App-0

Figure 13-28. 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 13-114
- 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

Note: The predefined BIRT reports and the Reports database and associated tables such as `APP_ACTIVITY_REPORT` are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

The MobileFirst installation contains a number of predefined BIRT reports. These reports are configurable XML files that are designed to retrieve and present data from the MobileFirst 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.

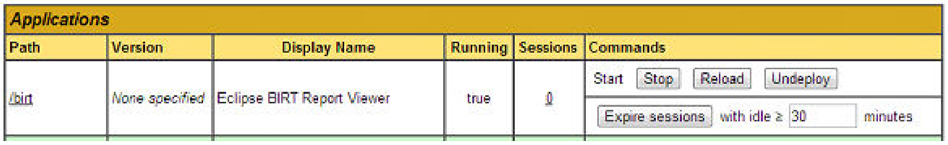
Procedure

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 IBM MobileFirst Platform 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:MaxPermSize=256m
```

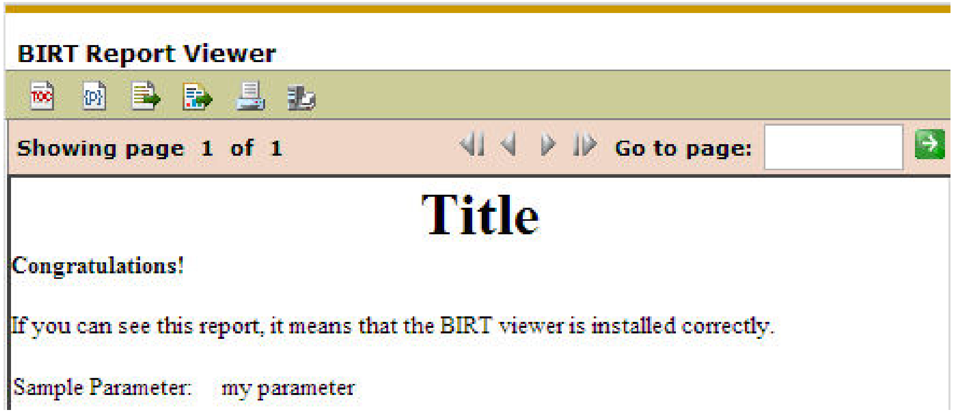
8. Restart your Tomcat.
9. Go to theTomcat manager application at http://your-server/manager/ to verify that the BIRT Reports application started.



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 minutes

10. Your BIRT Reports viewer application is accessible at http://your-server/birt/.
11. You can test the BIRT Reports installation by going to http://your-server/birt/frameset?__report=test.rptdesign&sample=my+parameter.



BIRT Report Viewer

Showing page 1 of 1

Title

Congratulations!

If you can see this report, it means that the BIRT viewer is installed correctly.

Sample Parameter: my parameter

Installing BIRT on WebSphere Application Server Liberty profile

Complete these steps to install Business Intelligence Reporting Tools on the WebSphere Application Server Liberty profile.

About this task

Note: The predefined BIRT reports and the Reports database and associated tables such as APP_ACTIVITY_REPORT are deprecated in IBM MobileFirst Platform

Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

Procedure

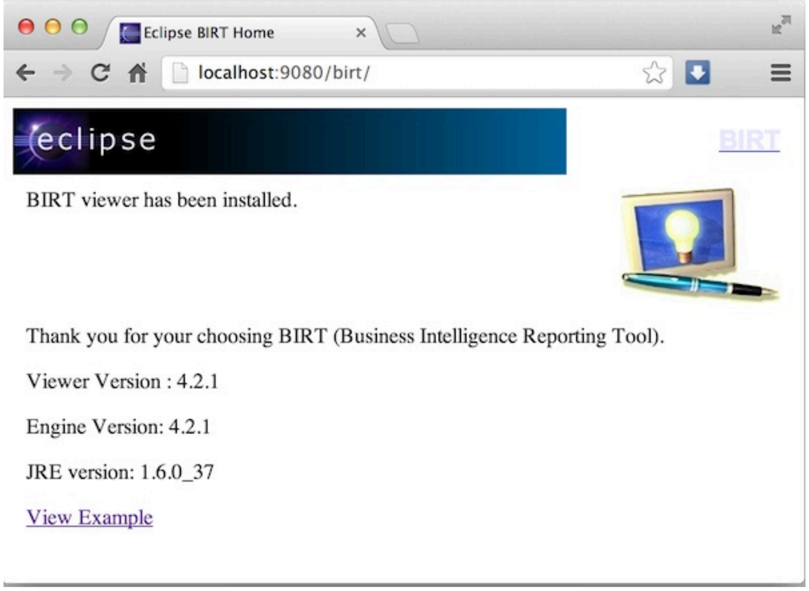
1. Verify that your WebSphere Application Server Liberty profile instance is not running.
2. Go to your WebSphere Application Server Liberty profile folder and create two folders as follows:
   - apps
   - libs
3. Locate the jdbc connector driver that you are using and copy it to the `libs` folder.
4. Download the latest release of BIRT run time from http://download.eclipse.org/birt/downloads/
5. Extract the downloaded file and go to the extracted folder.
6. Rename `WebViewerExample` folder to `birt.`
7. Go to the folder `birt\WEB-INF\lib` and delete the following files.
   - org.apache.xerces*.jar
   - org.apache.xml.resolver*.jar
   - org.apache.xml.serializer*.jar

   Set up the BIRT Viewer application on a Liberty instance by following these steps.
8. Copy the `birt` folder to `{your-liberty-instance}\usr\servers\{your-server-name}\apps\`
9. Update the `server.xml` file of your Liberty server profile.
10. Make sure that the JSP feature is enabled.
11. Add an application definition.
12. Add `classloader` definition with a `privateLibrary` definition that is configured to point to your JDBC connector driver.

```
<server description="new server">
  <featureManager>
    <feature>jsp-2.2</feature>
  </featureManager>

  <httpEndpoint id="defaultHttpEndpoint"
               host="*"
               httpPort="9080"
               httpsPort="9443" />

  <application id="birt"
              name="birt"
              type="war"
              location="${server.config.dir}/apps/birt"
              context-root="/birt">
    <classloader delegation="parentLast">
      <privateLibrary>
        <fileset dir="${server.config.dir}/libs"
                includes="mysql-connector*.jar" />
      </privateLibrary>
    </classloader>
  </application>
</server>
```

13. Start your Liberty instance.

14. Browse to `http://server:port/birt`. The BIRT Viewer landing page opens.



BIRT viewer has been installed.

Thank you for your choosing BIRT (Business Intelligence Reporting Tool).

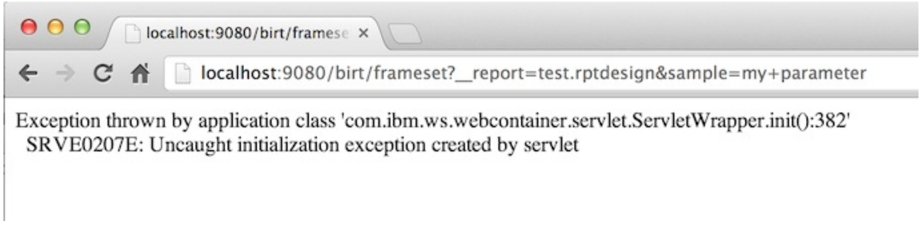
Viewer Version : 4.2.1

Engine Version: 4.2.1

JRE version: 1.6.0_37

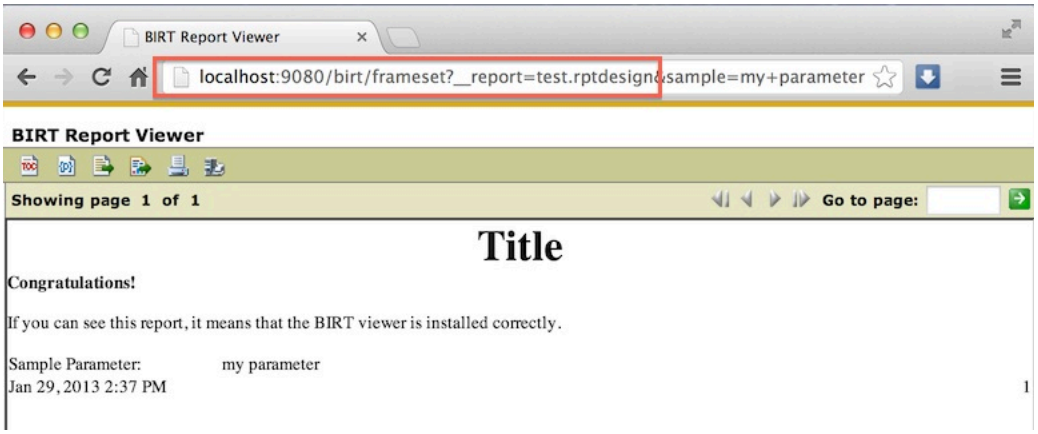
View Example

15. Click **View Example** link.
16. If you see the following error message, refresh your page.


localhost:9080/birt/framese
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter

Exception thrown by application class 'com.ibm.ws.webcontainer.servlet.ServletWrapper.init():382'
SRVE0207E: Uncaught initialization exception created by servlet

17. The BIRT Viewer sample report appears.


BIRT Report Viewer
localhost:9080/birt/frameset?__report=test.rptdesign&sample=my+parameter
BIRT Report Viewer
Showing page 1 of 1
Go to page:

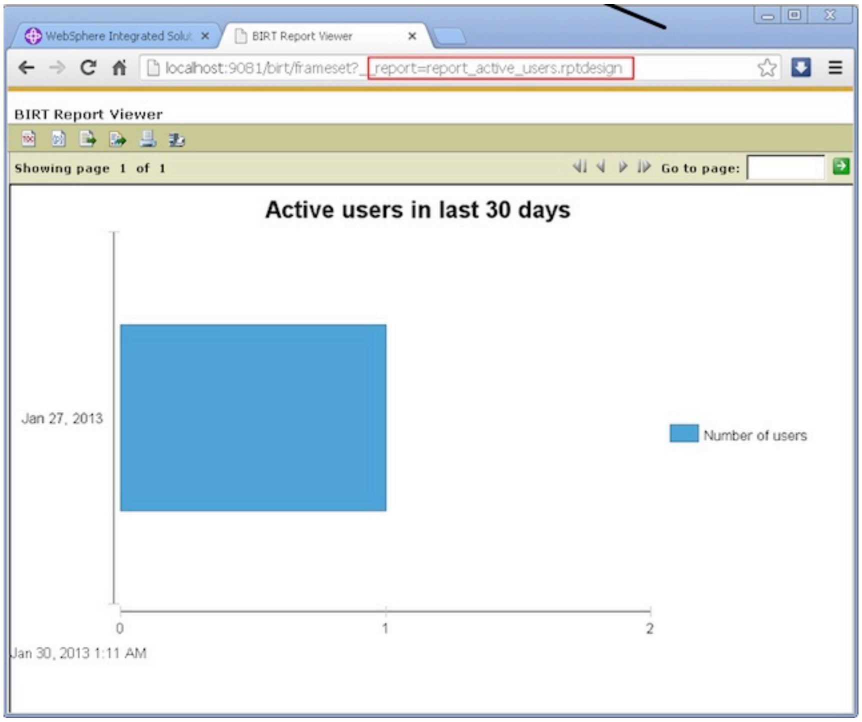
Title

Congratulations!

If you can see this report, it means that the BIRT viewer is installed correctly.

Sample Parameter: my parameter
Jan 29, 2013 2:37 PM 1

Note `test.rptdesign` in the page URL. You can replace this text with the name of other **rptdesign** files, as shown here for example:


WebSphere Integrated Solu
BIRT Report Viewer
localhost:9081/birt/frameset?_report=report_active_users.rptdesign
BIRT Report Viewer
Showing page 1 of 1
Go to page:
Active users in last 30 days
Jan 27, 2013
Number of users
0
1
2
Jan 30, 2013 1:11 AM

Installing BIRT on WebSphere Application Server full profile

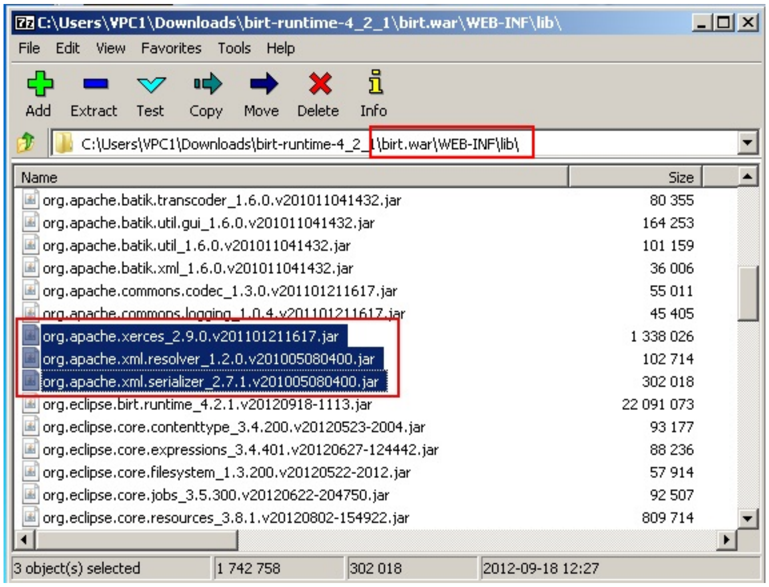
Complete these steps to install Business Intelligence Reporting Tools (BIRT) on WebSphere Application Server full profile.

About this task

Note: The predefined BIRT reports and the Reports database and associated tables such as `APP_ACTIVITY_REPORT` are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

Procedure

1. Download the BIRT package and extract the contents.
2. From the folder `birt-runtime-version\WebViewerExample\WEB-INF\lib`, delete (or remove) the following packages:
   - `org.apache.xerces.jar`
   - `org.apache.resolver.jar`
   - `org.apache.serializer.jar`


C:\Users\VPC1\Downloads\birt-runtime-4_2_1\birt.war\WEB-INF\lib\
File
Edit
View
Favorites
Tools
Help
Add
Extract
Test
Copy
Move
Delete
Info

C:\Users\VPC1\Downloads\birt-runtime-4_2_1\birt.war\WEB-INF\lib\

Name	Size
org.apache.batik.transcoder_1.6.0.v201011041432.jar	80 355
org.apache.batik.util.gui_1.6.0.v201011041432.jar	164 253
org.apache.batik.util_1.6.0.v201011041432.jar	101 159
org.apache.batik.xml_1.6.0.v201011041432.jar	36 006
org.apache.commons.codec_1.3.0.v201101211617.jar	55 011
org.apache.commons.logging_1.0.4.v201101211617.jar	45 405
org.apache.xerces_2.9.0.v201101211617.jar	1 338 026
org.apache.xml.resolver_1.2.0.v201005080400.jar	102 714
org.apache.xml.serializer_2.7.1.v201005080400.jar	302 018
org.eclipse.birt.runtime_4.2.1.v20120918-1113.jar	22 091 073
org.eclipse.core.contenttype_3.4.200.v20120523-2004.jar	93 177
org.eclipse.core.expressions_3.4.401.v20120627-124442.jar	88 236
org.eclipse.core.filesystem_1.3.200.v20120522-2012.jar	57 914
org.eclipse.core.jobs_3.5.300.v20120622-204750.jar	92 507
org.eclipse.core.resources_3.8.1.v20120802-154922.jar	809 714

3 object(s) selected | 1 742 758 | 302 018 | 2012-09-18 12:27

Figure 13-29. 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**.
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

Note: The predefined BIRT reports and the Reports database and associated tables such as `APP_ACTIVITY_REPORT` are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use “Operational analytics” on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

To use BIRT reports, you must update them with your web application server settings and install them in your server web applications folder. The easiest way to do this is to specify a <reports> element in the Ant script that invokes the <configureapplicationserver> Ant task.

Procedure

1. Ensure that the <configureapplicationserver> invocation has the inner element <reports todir="*web applications directory*"/>. See "Ant tasks for installation of MobileFirst runtime environments" on page 15-32 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

Note: The predefined BIRT reports and the Reports database and associated tables such as APP_ACTIVITY_REPORT are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

Before using the BIRT Viewer application to see predefined reports, you must edit them to adjust the reports database settings, and then copy the reports to a specific folder on the application server.

Procedure

1. Go to your IBM MobileFirst Platform 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 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.

Note: The predefined BIRT reports and the Reports database and associated tables such as APP_ACTIVITY_REPORT are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

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 MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS can provide you with a more integrated interface, for both IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Eclipse

You can install BIRT in the instance of Eclipse on which IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS, follow these steps:

Procedure

1. Click **Help** > **Install new software**
2. In **Work with...**, select `http://download.eclipse.com/release/`*`Eclipse_instance`*
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 MobileFirst 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 MobileFirst reports database
15. Select **Driver Class** and adjust the rest of your database settings
16. Click **Test Connection...** to validate that database settings are correct.

Viewing BIRT reports in Eclipse

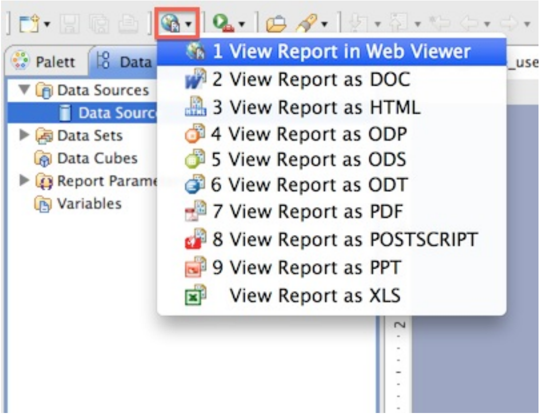
With BIRT installed in Eclipse, you can view reports through the Eclipse interface.

About this task

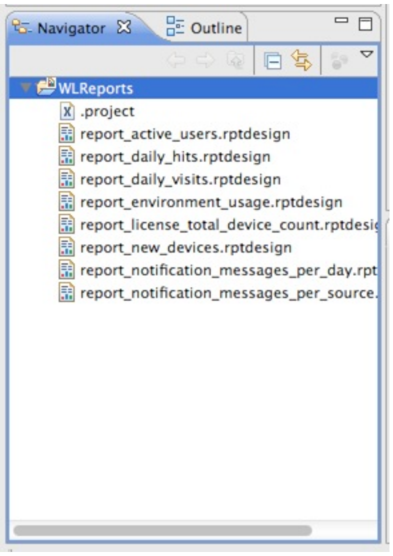
To view BIRT reports in Eclipse, follow these steps:

Procedure

1. Click the black arrow next to **View Report**.



2. Select the output format for your report
3. View the report.

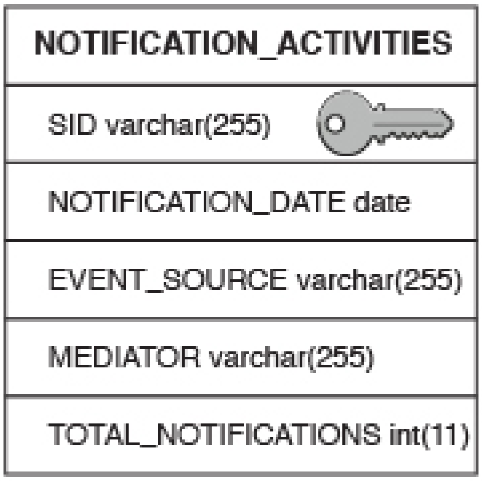


Navigator
Outline
WLReports
.project
report_active_users.rptdesign
report_daily_hits.rptdesign
report_daily_visits.rptdesign
report_environment_usage.rptdesign
report_license_total_device_count.rptdesi
report_new_devices.rptdesign
report_notification_messages_per_day.rpt
report_notification_messages_per_source.

Notification reports database schema

IBM MobileFirst Platform Foundation for iOS uses a database schema to store the notification reports data derived from the raw data.

Note: The predefined BIRT reports and the Reports database and associated tables such as `NOTIFICATION_ACTIVITIES` are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.



NOTIFICATION_ACTIVITIES

SID varchar(255)

NOTIFICATION_DATE date

EVENT_SOURCE varchar(255)

MEDIATOR varchar(255)

TOTAL_NOTIFICATIONS int(11)

Figure 13-30. 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.

Manually creating and configuring the reports database

You can manually create and configure the reports databases.

The reports database WLREPORT is deprecated since IBM MobileFirst Platform Foundation for iOS V7.0.0. Use “Operational analytics” on page 13-6 instead.

If needed, you can still manually configure your reports database. See the topics in the section Creating and configuring the databases manually, and substitute the parameters and instructions for the runtime database with the instructions in the following topics, which are specific to the reports database.

Manually configuring the DB2 reports database

You can manually set up the reports database for DB2, and configure it according to your application server.

About this task

Set up your DB2 reports database and configure your application server. Depending on your application server, follow the steps for WebSphere Application Server Liberty profile, WebSphere Application Server full profile, or Apache Tomcat.

Procedure

- Set up your DB2 reports database manually.

```
CREATE DATABASE WLREPORT COLLATE USING SYSTEM PAGESIZE 32768
CONNECT TO WLREPORT
GRANT CONNECT ON DATABASE TO USER worklight
QUIT
```

```
db2 CONNECT TO WLREPORT USER worklight USING password
db2 SET CURRENT SCHEMA = 'WLRESCHM'
db2 -vf product_install_dir/WorklightServer/databases/create-worklightreports-db2.sql -t
```

Steps for WebSphere Application Server Liberty profile.

- Configure Liberty profile for DB2 manually.

```
<!-- Declare the 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>
```

Steps for WebSphere Application Server full profile.

- Configure WebSphere Application Server for DB2 manually.
  1. Set the **Data source name** to **Worklight Reports Database**.
  2. Set **JNDI Name** to **jdbc/WorklightReportsDS**.
  3. Click **Next**.
  4. Enter properties for the data source, for example:
     - **Database Name**: WLREPORT

Steps for Apache Tomcat.

- Configure Apache Tomcat for DB2 manually.

```
<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;"/>
```

Manually configuring the Apache Derby reports database

You can manually set up the reports database for Apache Derby, and configure it according to your application server.

About this task

First, set up your Apache Derby reports database. Then, configure your application server. According to your application server, follow the steps for WebSphere Application Server Liberty profile, or WebSphere Application Server full profile, or Apache Tomcat.

Procedure

- Set up your Apache Derby reports database manually.

```
connect 'jdbc:derby:WLREPORT;user=WORKLIGHT;create=true';
run 'product_install_dir/WorklightServer/databases/create-worklightreports-derby.sql';
```

Steps for WebSphere Application Server Liberty profile.

- Configure Liberty profile for Derby manually.

```
<!-- Declare the reports database -->
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false" statementCacheSize=
  <jdbcDriver libraryRef="derbyLib" javax.sql.ConnectionPoolDataSource="org.apache.derby.jdbc.Embe
  <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>
```

Steps for WebSphere Application Server full profile.

- Configure WebSphere Application Server for Derby manually.
  1. Set **Data source name** to **Worklight Reports Database**.
  2. Set **JNDI name** to `jdbc/WorklightReportsDS`.

Steps for Apache Tomcat

- Configure Apache Tomcat for Derby manually.

```
<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"/>
```

Manually configuring the MySQL reports database

You can manually set up the reports database for MySQL, and configure it according to your application server.

About this task

First, set up your MySQL reports database. Then, configure your application server. According to your application server, follow the steps for WebSphere Application Server Liberty profile, or WebSphere Application Server full profile, or Apache Tomcat.

Procedure

- Set up your MySQL reports database manually.

```
CREATE DATABASE WLREPORT CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL PRIVILEGES ON WLREPORT.* TO 'worklight'@'Worklight-host' IDENTIFIED BY 'worklight';
GRANT ALL PRIVILEGES ON WLREPORT.* TO 'worklight'@'localhost' IDENTIFIED BY 'worklight';
FLUSH PRIVILEGES;

USE WLREPORT; SOURCE product_install_dir/WorklightServer/databases/create-worklightreports-mysq
```

Steps for WebSphere Application Server Liberty profile

- Configure Liberty profile for MySQL manually.

```
<!-- Declare the reports database -->
<dataSource jndiName="worklight/jdbc/WorklightReportsDS" transactional="false">
  <jdbcDriver libraryRef="MySQLLib"/>
  <properties databaseName="WLREPORT"
              serverName="mysqlserver" portNumber="3306"
              user="worklight" password="password"/>
</dataSource>
```

Steps for WebSphere Application Server full profile

- Configure WebSphere Application Server for MySQL manually.
  1. Set **JNDI Name** to `jdbc/WorklightReportsDS`.
  2. Set the custom properties of the new data source.

     ```
     databaseName = WLREPORT
     ```

Steps for Apache Tomcat

- Configure Apache Tomcat for MySQL manually.

```
<Resource name="jdbc/WorklightReportsDS"
                    auth="Container"
                    type="javax.sql.DataSource"
                    maxActive="100"
                    maxIdle="30"
                    maxWait="10000"
                    username="worklight"
                    password="worklight"
                    driverClassName="com.mysql.jdbc.Driver"
                    url="jdbc:mysql://mysqlserver:3306/WLREPORT"/>
```

Manually configuring the Oracle reports database

You can manually set up the reports database for Oracle, and configure it according to your application server.

About this task

First, set up your Oracle reports database. Then, configure your application server. According to your application server, follow the steps for WebSphere Application Server Liberty profile, or WebSphere Application Server full profile, or Apache Tomcat.

Procedure

- Set up your Oracle reports database manually.

```
CONNECT SYSTEM/SYSTEM_password@ORCL
CREATE USER WORKLIGHTREPORTS IDENTIFIED BY WORKLIGHTREPORTS_password DEFAULT TABLESPACE USERS QUOT
GRANT CREATE SESSION, CREATE SEQUENCE, CREATE TABLE TO WORKLIGHTREPORTS;
DISCONNECT;
```

```
CONNECT WORKLIGHTREPORTS/<WORKLIGHTREPORTS_password>@ORCL
@product_install_dir/WorklightServer/databases/create-worklightreports-oracle.sql
DISCONNECT;
```

Steps for WebSphere Application Server Liberty profile

- Configure Liberty profile for Oracle manually.

```
<!-- Declare the 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>
```

Steps for WebSphere Application Server full profile

- Configure WebSphere Application Server for Oracle manually.
  1. Set **JNDI name** to `jdbc/WorklightReportsDS`.
  2. Set **user = WORKLIGHTREPORTS**.

Steps for Apache Tomcat

- Configure Apache Tomcat for Oracle manually.

```
<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"/>
```

Upgrading the reports database

If you still use the reports database and want to upgrade it to MobileFirst V7.1.0, you can upgrade with Ant tasks or manually.

The reports database `WLREPORT` is deprecated since IBM MobileFirst Platform Foundation for iOS V7.0.0. Use "Operational analytics" on page 13-6 instead. If needed, you can still have a reports database and upgrade it to V7.1.0. In this case, you have two options to upgrade. You can upgrade the reports database with Ant tasks, or manually.

Using Ant tasks to upgrade the reports database

You can choose to use Ant tasks to upgrade your reports database to IBM MobileFirst Platform Foundation for iOS V7.1.0. This is the preferred way.

Procedure

1. See the procedure at "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21 to identify and specify the correct properties of the Ant sample that you use for your application server and your database, for example `configure-liberty-db2.xml` if you have WebSphere Application Server Liberty profile and a DB2 database.
2. After you identify and specify the properties, add the following properties about the reports database to your Ant sample, according to your database type.
   - For DB2:
     ```
     <property name="database.db2.worklightreports.password" value="************"/>
     <property name="database.db2.worklightreports.dbname" value="****UPDATE**** - Database name
     <property name="database.db2.worklightreports.schema" value="****UPDATE**** - Database Schem
     <property name="database.db2.worklightreports.username" value="****UPDATE**** - DB2 user nam
     ```
   - For MySQL:
     ```
     <property name="database.mysql.worklightreports.password" value="************"/>
     <property name="database.mysql.worklightreports.dbname" value="****UPDATE**** - Database nam
     <property name="database.mysql.worklightreports.username" value="****UPDATE**** - MySQL user
     ```
   - For Oracle:
     ```
     <property name="database.oracle.worklightreports.password" value="************"/>
     <property name="database.oracle.worklightreports.dbname" value="****UPDATE**** - Database na
     <property name="database.oracle.worklightreports.username" value="****UPDATE**** - User name
     ```
3. Modify the database target according to your database type. Find the `<target name="databases">` target in your Ant sample, and add the following element to that target:
   - For DB2:
     ```
     <configuredatabase kind="WorklightReports">
       <db2 database="${database.db2.worklightreports.dbname}"
            server="${database.db2.host}"
            instance="${database.db2.instance}"
            user="${database.db2.worklightreports.username}"
            port="${database.db2.port}"
            schema="${database.db2.worklightreports.schema}"
            password="${database.db2.worklightreports.password}">
       </db2>
       <driverclasspath>
         <fileset dir="${database.db2.driver.dir}">
           <include name="db2jcc4.jar"/>
           <include name="db2jcc_license_*.jar"/>
         </fileset>
       </driverclasspath>
     </configuredatabase>
     ```
   - For MySQL:

```
<configuredatabase kind="WorklightReports">
  <mysql database="${database.mysql.worklightreports.dbname}"
         server="${database.mysql.host}"
         port="${database.mysql.port}"
         user="${database.mysql.worklightreports.username}"
         password="${database.mysql.worklightreports.password}">
  </mysql>
  <driverclasspath>
    <pathelement location="${database.mysql.driver}"/>
  </driverclasspath>
</configuredatabase>
```

- For Oracle:

```
<configuredatabase kind="WorklightReports">
  <oracle database="${database.oracle.worklightreports.dbname}"
          server="${database.oracle.host}"
          port="${database.oracle.port}"
          user="${database.oracle.worklightreports.username}"
          password="${database.oracle.worklightreports.password}">
  </oracle>
  <driverclasspath>
    <pathelement location="${database.oracle.driver}"/>
  </driverclasspath>
</configuredatabase>
```

4. Modify the `minimal-update` target. Find the `<target name="minimal-update">` target and the `<updateapplicationserver>` element, and change the `useWorklightReports` property to `true`:

```
<target name="minimal-update">
  <updateapplicationserver useWorklightReports="true"
contextroot="${worklight.contextroot}">
```

5. According to your database type, add the following element to your `<target name="minimal-update">` target:

- For DB2:

```
<database kind="WorklightReports">
  <db2 database="${database.db2.worklightreports.dbname}"
       server="${database.db2.host}"
       user="${database.db2.worklightreports.username}"
       port="${database.db2.port}"
       schema="${database.db2.worklightreports.schema}"
       password="${database.db2.worklightreports.password}">
  </db2>
  <driverclasspath>
    <fileset dir="${database.db2.driver.dir}">
      <include name="db2jcc4.jar"/>
      <include name="db2jcc_license_*.jar"/>
    </fileset>
  </driverclasspath>
</database>
```

- For MySQL:

```
<database kind="WorklightReports">
  <mysql database="${database.mysql.worklightreports.dbname}"
         server="${database.mysql.host}"
         port="${database.mysql.port}"
         user="${database.mysql.worklightreports.username}"
         password="${database.mysql.worklightreports.password}">
  </mysql>
  <driverclasspath>
    <pathelement location="${database.mysql.driver}"/>
  </driverclasspath>
</database>
```

- For Oracle:

```
<database kind="WorklightReports">
  <oracle database="${database.oracle.worklightreports.dbname}"
          server="${database.oracle.host}"
          port="${database.oracle.port}"
          user="${database.oracle.worklightreports.username}"
          password="${database.oracle.worklightreports.password}"/>
  <driverclasspath>
    <pathelement location="${database.oracle.driver}"/>
  </driverclasspath>
</database>
```

What to do next

After you complete these steps, you can resume the upgrade procedure in "Identify the MobileFirst WAR file and prepare the Ant deployment script" on page 7-21.

Manually upgrading the reports database

You can choose to manually upgrade your reports database to IBM MobileFirst Platform Foundation for iOS V7.1.0. You can use this option if you are not able to use the Ant tasks.

Procedure

- You must follow the procedure at "Manually upgrading the MobileFirst Server V7.1.0 databases" on page 7-62.
- Verify that a data source for your reports database is declared in your application server. For more information about how to add this data source, see "Manually creating and configuring the reports database" on page 13-117.
- Enable your reports database by setting the JNDI property `reports.exportRawData` to `true`. For more information, see "JNDI environment entries for MobileFirst projects in production" on page 11-67.

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 MobileFirst Operations Console.

The MobileFirst Server runtime tracks devices that access your mobile infrastructure by the MobileFirst 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 MobileFirst Operations Console now provides a view into this mapping of user to devices through the apps that are used to access your MobileFirst 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 MobileFirst App that is installed on the device from connecting to the MobileFirst 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 MobileFirst Server, administrators can disable access for all MobileFirst applications on that device.

In some cases, it might not be desirable to block access for every MobileFirst application that is installed on the device. MobileFirst 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 MobileFirst application requires a certificate from the user to authenticate, the serial number of the certificate is recorded on the MobileFirst Server. In addition to viewing each application installed on a device, the certificate serial number can also be viewed in the MobileFirst Operations 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.

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Server in 90 days) - configurable
- disabled

When a MobileFirst application from a device attempts to connect through the MobileFirst 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 MobileFirst Operations 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 MobileFirst Server tracks the devices that access the system as part of the core runtime 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 MobileFirst 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:

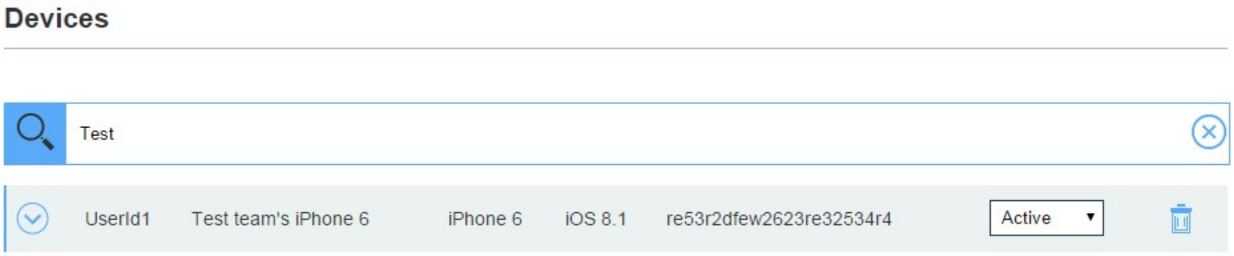
- Disable the specific device, marking the state as lost or stolen so that access from any of the apps on that device is blocked.
- Re-enable a disabled device so that access from the device to the MobileFirst Server is allowed.
- Disable a specific app, marking the state as disabled so that access from 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 MobileFirst Server is allowed.

Device access management in the MobileFirst Operations Console

In the MobileFirst Operations Console, administrators can search for devices that access the MobileFirst Server and can manage 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)..

When a valid device is found, all devices that match the user ID or friendly name are listed.


Devices

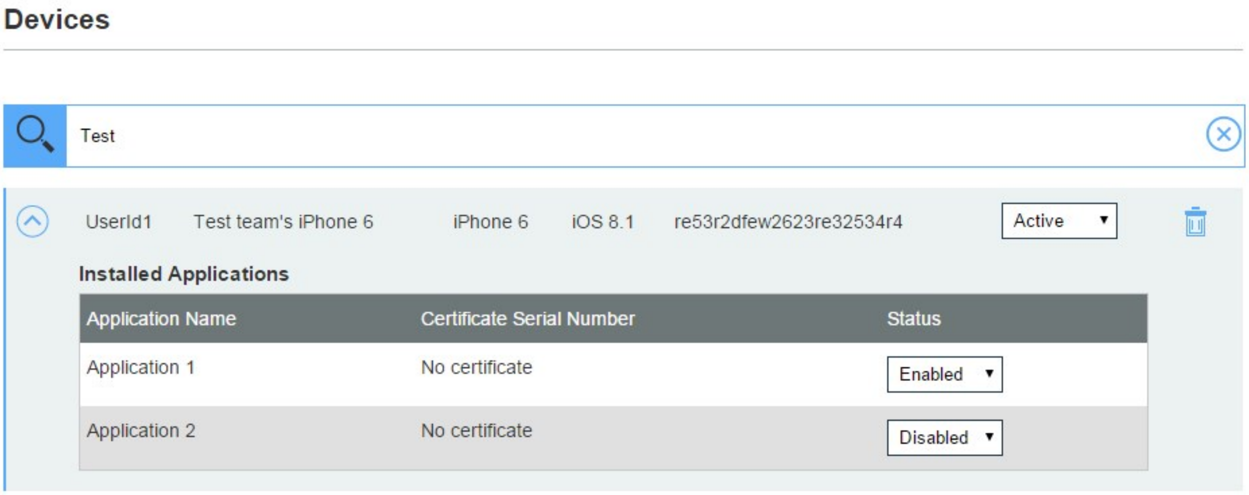
Test

UserId1	Test team's iPhone 6	iPhone 6	iOS 8.1	re53r2dfew2623re32534r4	Active ▾

Figure 13-31. 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 MobileFirst 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 MobileFirst Server and its status is changed to “Active”. For more information about licensing, see “License Tracking report” on page 13-130.

Clicking the **expand arrow** icon in the column shows a list of all applications that this device accessed.



Devices

Test

UserId1 Test team's iPhone 6 iPhone 6 iOS 8.1 re53r2dfew2623re32534r4 Active ▾

Installed Applications

Application Name	Certificate Serial Number	Status
Application 1	No certificate	Enabled ▾
Application 2	No certificate	Disabled ▾

Figure 13-32. 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 MobileFirst Server for this device.

Enabling the device access management features

All devices that access the MobileFirst Server are recorded in the runtime database without any additional configurations. However, IBM MobileFirst Platform Foundation for iOS does not enforce the device access settings that are set from the MobileFirst Operations Console unless you enable a property on the MobileFirst Server.

About this task

More processing is required on the MobileFirst 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 MobileFirst 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 runtime 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 8-223.

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 runtime database.

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 IBM MobileFirst Platform Foundation for iOS save information about a device when it accesses the server?
2. Does IBM MobileFirst Platform Foundation for iOS enforce access rights when a device tries to access the server?

Saving device information

The MobileFirst administrator can control whether the server saves device information to the internal database when a device connects to the MobileFirst 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 MobileFirst 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 13-128.

Enforcing access rights

The MobileFirst Server tries to save the device information only on the first request of a session from the device. However, IBM MobileFirst Platform Foundation for iOS 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 MobileFirst 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 MobileFirst Operations Console. However, they cannot block access from the device to the MobileFirst 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', 3)
```

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 MobileFirst Server serves thousands of devices that use multiple applications that are hosted by the server. Devices can be deleted from the runtime database in the MobileFirst Operations 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 administrators must not delete data from these rows if the action of deleting the data affects licensing.

License tracking

IBM MobileFirst Platform Foundation for iOS 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 IBM MobileFirst Platform Foundation for iOS, which tracks metrics relevant to the licensing policy such as active client device, addressable devices, and installed apps.

This information helps determine if the current usage of IBM MobileFirst Platform Foundation for iOS 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, MobileFirst administrators can decommission devices that are no longer accessing the IBM MobileFirst Platform. This situation might arise if an employee has left the company, for example. Addressable devices are only counted for the current calendar month. Devices that did not use an application in the current calendar month are not counted.

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 IBM MobileFirst Platform Operations 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 MobileFirst platform. This data can be displayed in the License Tracking report, which is accessed from the IBM MobileFirst Platform Operations 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 IBM MobileFirst Platform 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 IBM MobileFirst Platform Foundation for iOS. For performance reasons, you can disable this flag when IBM MobileFirst Platform Foundation for iOS is neither licensed by Client Device nor by Addressable Device. When device tracking is disabled, the license reports are also disabled and no license metrics are generated. In that case, only ILMT records for Application count are generated.

For more information about specifying JNDI properties, see Configuring an IBM MobileFirst project in production by using JNDI environment entries.

The decommissioning task is run daily, as a MobileFirst 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 MobileFirst Operations Console. For more information, see “License Tracking report” on page 13-130.

License Tracking report

IBM MobileFirst Platform Foundation for iOS provides a report that shows how many client devices accessed 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 IBM MobileFirst Platform Server
- The number of client devices, both active and decommissioned
- The number of addressable devices in the current calendar month
- 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.

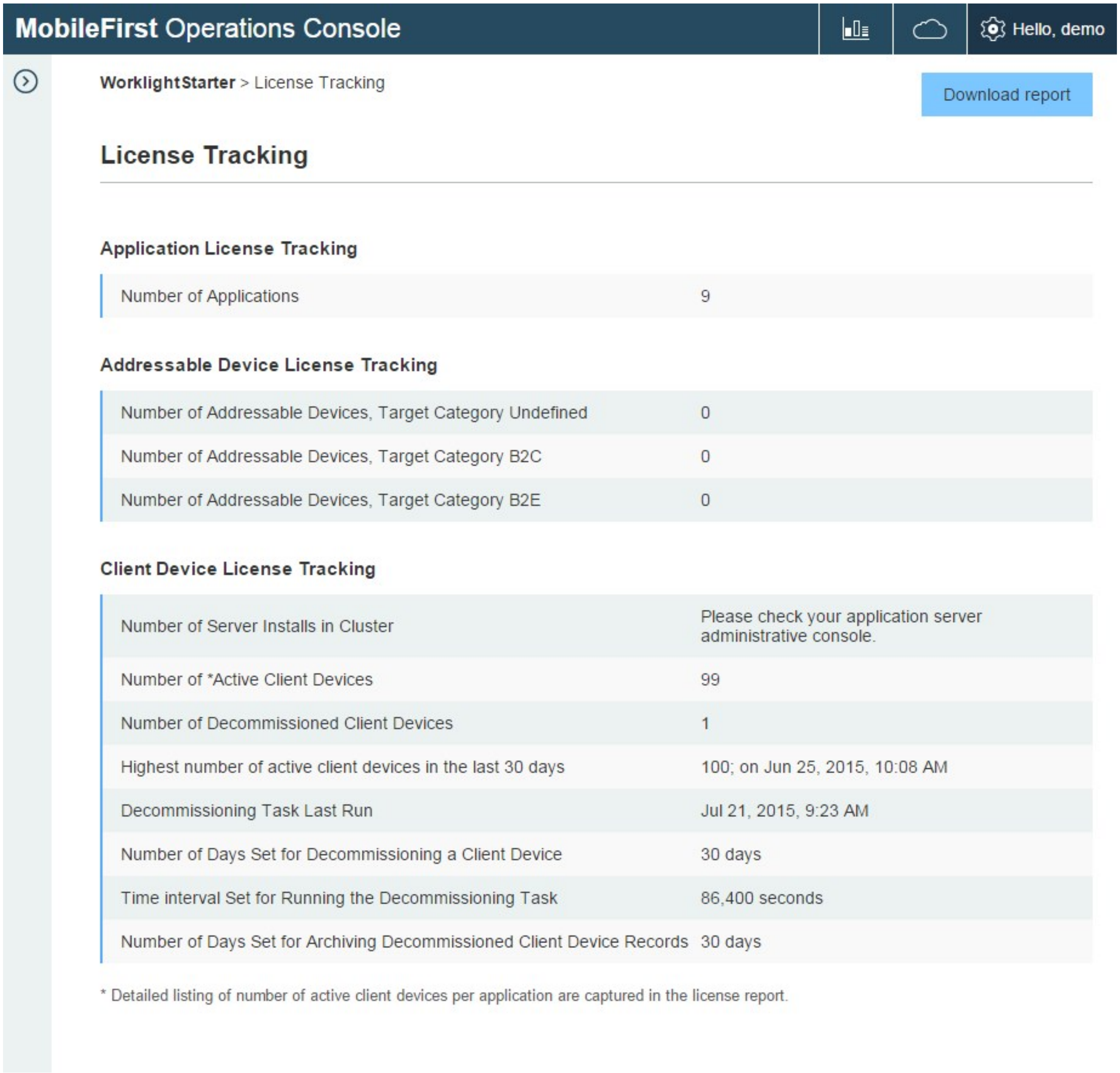
Administrators might want to analyze data further. For this purpose, the number of active client devices per application, the generated report details, the addressable devices per application in the last 12 months, and a historical listing of license metrics are captured in a CSV file that can be downloaded 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 **License tracking** link in a runtime environment in the IBM MobileFirst Platform Operations Console.


MobileFirst Operations Console
Hello, demo
WorklightStarter > License Tracking
Download report

License Tracking

Application License Tracking

Number of Applications	9

Addressable Device License Tracking

Number of Addressable Devices, Target Category Undefined	0
Number of Addressable Devices, Target Category B2C	0
Number of Addressable Devices, Target Category B2E	0

Client Device License Tracking

Number of Server Installs in Cluster	Please check your application server administrative console.
Number of *Active Client Devices	99
Number of Decommissioned Client Devices	1
Highest number of active client devices in the last 30 days	100; on Jun 25, 2015, 10:08 AM
Decommissioning Task Last Run	Jul 21, 2015, 9:23 AM
Number of Days Set for Decommissioning a Client Device	30 days
Time interval Set for Running the Decommissioning Task	86,400 seconds
Number of Days Set for Archiving Decommissioned Client Device Records	30 days

* Detailed listing of number of active client devices per application are captured in the license report.

Figure 13-33. License tracking information for applications, devices, and decommissioning

To obtain a CSV file from the License Tracking report, click **Download report**.

Integration with IBM License Metric Tool

IBM MobileFirst Platform Foundation for iOS generates IBM Software License Metric Tag (SLMT) files. Versions of IBM License Metric Tool that support IBM Software License Metric Tag can generate License Consumption Reports. Read this section to interpret these reports for MobileFirst Server, and to configure the generation of the IBM Software License Metric Tag files.

If you have not installed a version of IBM License Metric Tool that supports IBM Software License Metric Tag, you can review the license usage with the License Tracking reports of MobileFirst Operations Console. For more information, see “License Tracking report” on page 13-130.

Each instance of a running MobileFirst runtime environment generates an IBM Software License Metric Tag file. The metrics monitored are `CLIENT_DEVICE` and `APPLICATION`. Their values are refreshed every 24 hours.

About the `CLIENT_DEVICE` metric

The `CLIENT_DEVICE` metric can have the following subtypes:

Active Devices
The number of client devices that used the MobileFirst runtime environment, or another MobileFirst runtime instance belonging to the same cluster or server farm, and that were not decommissioned. For more information about decommissioned devices, see "Configuring your license tracking details" on page 13-129.

Inactive Devices
The number of client devices that used the MobileFirst runtime environment, or another MobileFirst runtime instance belonging to the same cluster or server farm, and that were decommissioned. For more information about decommissioned devices, see "Configuring your license tracking details" on page 13-129.

The following cases are specific:

- If the decommissioning period of the device is set to a small period, the subtype "Inactive Devices" is replaced by the subtype "Active or Inactive Devices".
- If device tracking was disabled, only one entry is generated for `CLIENT_DEVICE`, with the value 0, and the metric subtype "Device Tracking Disabled".
- If the MobileFirst runtime environment is running in a development server, and device tracking is not disabled, only one entry is generated for `CLIENT_DEVICE`. This entry has the sum of active and decommissioned devices as its value, and "Development Server" as its metric subtype.

About the `APPLICATION` metric

The `APPLICATION` metric has no subtype unless the MobileFirst runtime environment is running in a development server.

The value reported for this metric is the number of applications that are deployed in the MobileFirst runtime environment. Each application is counted as one unit, whether it is a new application, an additional brand deployment, or an additional type of an existing application (for example native, hybrid, or web).

This number of applications is monitored even if **`wl.device.tracking.enabled`** is set to false.

The following cases are specific:

- If the MobileFirst runtime environment is running in a development server, the metric `Application` is reported with the subtype "Development Server".

About the `ADDRESSABLE_DEVICE` metric

The `ADDRESSABLE_DEVICE` metric has the following subtype:

- Application: `<applicationName>`, Category: `<targetCategory>`

The target category of an application is B2C, B2E, or UNDEFINED. To define the target category of an application, see the following topics according to your operating system:

- For a native iOS application, see "Application descriptor of native API applications" on page 8-7

The following cases are specific:

- If the decommissioning period of the device is set to less than 30 days, the warning "Short decommissioning period" is appended to the subtype.
- If device tracking was disabled, no addressable report is generated.
- If the MobileFirst runtime environment is running in a development server, and device tracking is not disabled, "Development Server" is appended to the subtype.

Configuring IBM License Metric Tool log files

By default, the IBM Software License Metric Tag files are in the following directories:

- On Windows: `%ProgramFiles%\ibm\common\slm`
- On UNIX and UNIX-like operating systems: `/var/ibm/common/slm`

If the directories are not writable, the files are created in the log directory of the application server that runs the MobileFirst runtime environment.

You can configure the location and management of those files with the following properties:

- `license.metric.logger.output.dir`: Location of the IBM Software License Metric Tag files
- `license.metric.logger.file.size`: Maximum size of an SLMT file before a rotation is performed. The default size is 1 MB.
- `license.metric.logger.file.number`: Maximum number of SLMT archive files to keep in rotations. The default number is 10.

To change the default values, you must create a Java property file, with the format `key=value`, and provide the path to the properties file through the `license_metric_logger_configuration` JVM property.

Related tasks:

"Configuring your license tracking details" on page 13-129
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 MobileFirst platform. This data can be displayed in the License Tracking report, which is accessed from the IBM MobileFirst Platform Operations Console.

Token license validation

IBM MobileFirst Platform Server validates licenses during various scenarios.

The various scenarios in which licenses are validated are:

On application deployment

Licenses are checked out only if sufficient number of tokens are available.

License validation might fail in the following cases:

- The Rational Common Licensing native library is not installed and configured.

- The JNDI properties for Administration Services are not configured.
- Rational License Key Server is not accessible.
- Sufficient tokens are not available.
- The license has expired.

On application undeployment

Licenses are checked in when an application is undeployed.

At server startup

The license is checked out for every deployed application only if enough number of tokens are available for all deployed applications.

License validation might fail in the following cases:

- The Rational Common Licensing native library is not installed and configured.
- The JNDI properties for Administration Services are not configured.
- Rational License Key Server is not accessible.
- Sufficient tokens are not available.
- The license has expired.

At server shutdown

The license is checked in for every deployed application, during a server shutdown.

Integrating with other IBM products

IBM MobileFirst Platform Foundation for iOS integrates with other IBM products.

You can find samples and more documentation about such integration for developers and administrators on the Integration page of the Developer Center website for IBM MobileFirst Platform.

Introduction to MobileFirst integration capabilities

As a developer or administrator, you can use IBM MobileFirst Platform Foundation for iOS integration capabilities to connect specific IBM products to existing back-end systems and other Internet or intranet sources.

IBM MobileFirst Platform Foundation provides capabilities to integrate with IBM Endpoint Manager for Mobile Devices and IBM WebSphere Cast Iron for enterprise and application security.

IBM MobileFirst Platform Foundation for iOS also provides a flexible authentication framework to support existing security requirements through authenticator or login modules. For more information, see “MobileFirst security framework” on page 8-188.

Figure 1 gives a high-level view of the topology context for an app on a device that connects to IBM MobileFirst Platform Foundation for iOS.

Figure 2 shows where other IBM products fit within the typical MobileFirst topology diagram in Figure 1.

Item	Description
A	App
D	Device
N	Network
I/i	Internet or intranet
MFP	IBM MobileFirst Platform Foundation for iOS
EBE	Existing back ends
I	Other Internet sources

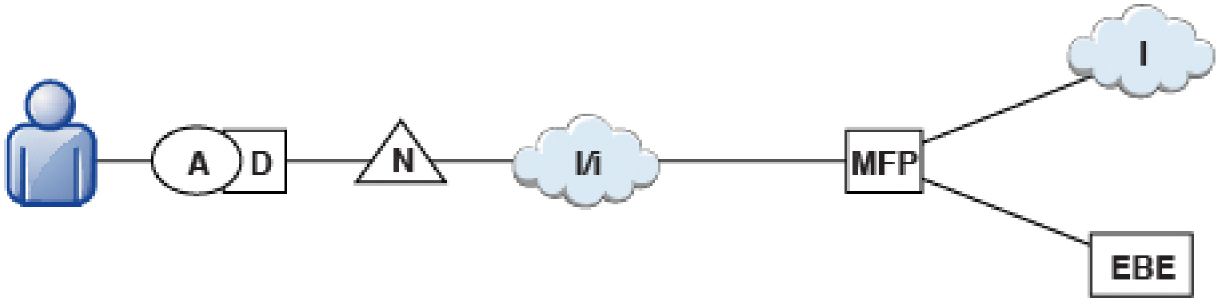


Figure 14-1. Overall Topology

Figure 14-2 shows where these products fit within the typical MobileFirst topology diagram in Figure 14-1.

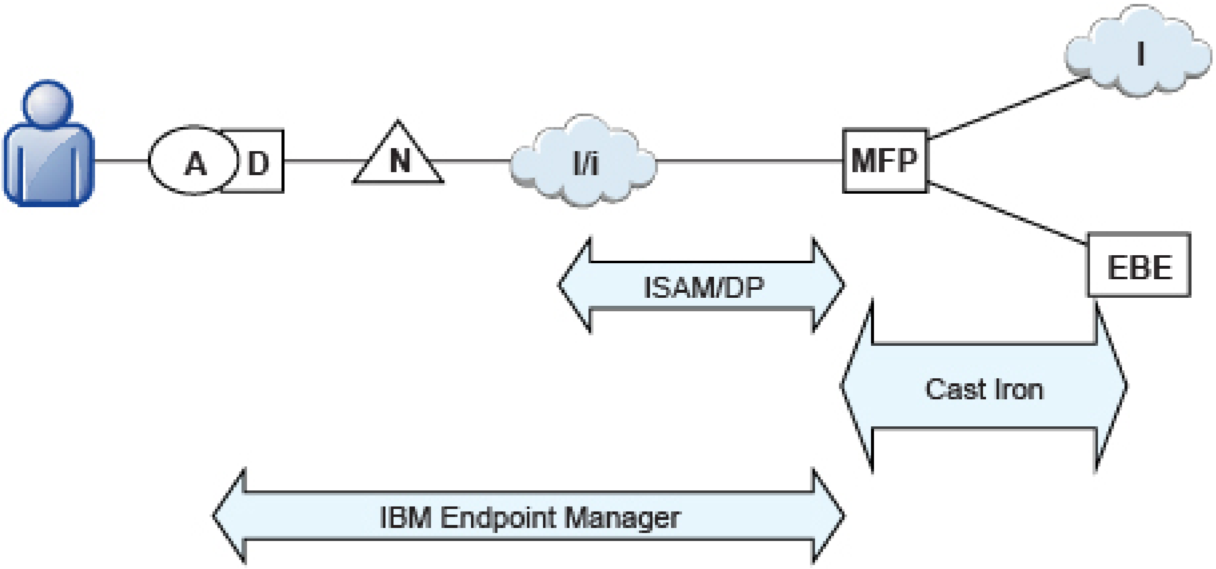


Figure 14-2. Integration Points

Integration with Cast Iron

You can use IBM WebSphere Cast Iron to enable enterprise connectivity within a MobileFirst environment.

IBM MobileFirst Platform Foundation for iOS supports the following adapters:

- SQL
- HTTP
- Cast Iron
- Java Message Service (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.

By using IBM WebSphere Cast Iron, companies can integrate applications, regardless of whether the applications are located on-premises or in public or private clouds. With WebSphere Cast Iron, you do not need any programming knowledge to integrate applications. 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 1 shows how the topology changes to reflect the use of Cast Iron.

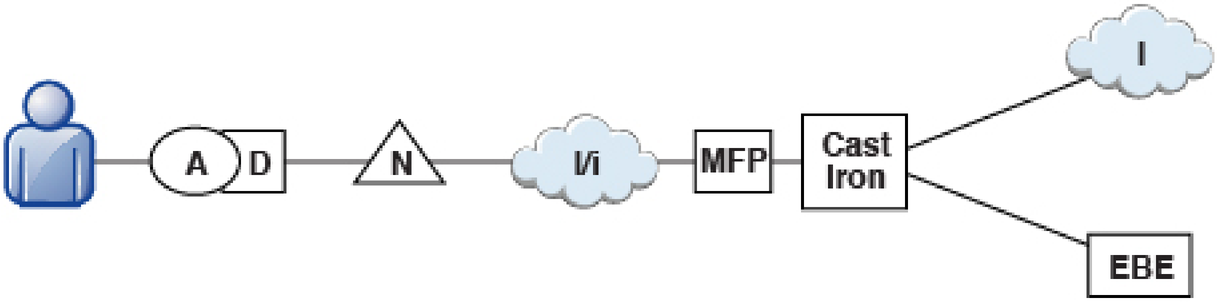


Figure 14-3. Integration with Cast Iron

For more information about Cast Iron adapters, see “Typical topologies of a MobileFirst instance in an extranet infrastructure” on page 6-299.

Integration and authentication with a reverse proxy

You can use a reverse proxy to enable enterprise connectivity within a MobileFirst environment and to provide authentication to IBM MobileFirst Platform Foundation for iOS.

General architecture

Reverse proxies typically front MobileFirst instances as part of the deployment, as shown in Figure 1, and follow the gateway pattern.

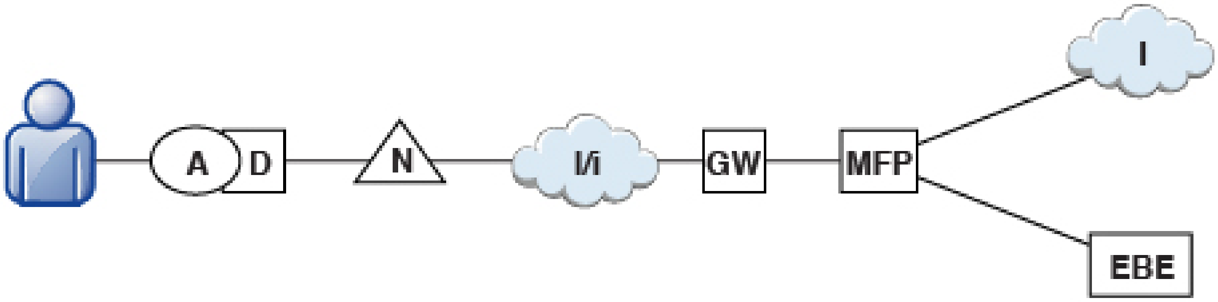


Figure 14-4. Integration with reverse proxy

The `MFP` icon represents an instance of MobileFirst Server. The `GW` icon represents a reverse-proxy gateway, such as IBM WebSphere DataPower or IBM Security Access Manager. In addition to protecting MobileFirst resources from the Internet, the reverse proxy provides termination of HTTPS (SSL) connections and authentication. The reverse proxy can also act as a policy enforcement point (PEP).

When a gateway is used, an application (A) on a device (D) uses the public URI that is advertised by the gateway instead of the internal MobileFirst Server URI. The public URI can be exposed as a setting within the application, or can be built in during promotion of the application to production, before the application is published to public or private application stores.

Authentication at the gateway

If authentication ends at the gateway, IBM MobileFirst Platform Foundation for iOS 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, you can configure IBM MobileFirst Platform Foundation for iOS to use the user identity from one of these mechanisms and establish a successful log in. Figure 14-5 shows a typical authentication flow for this gateway topology.

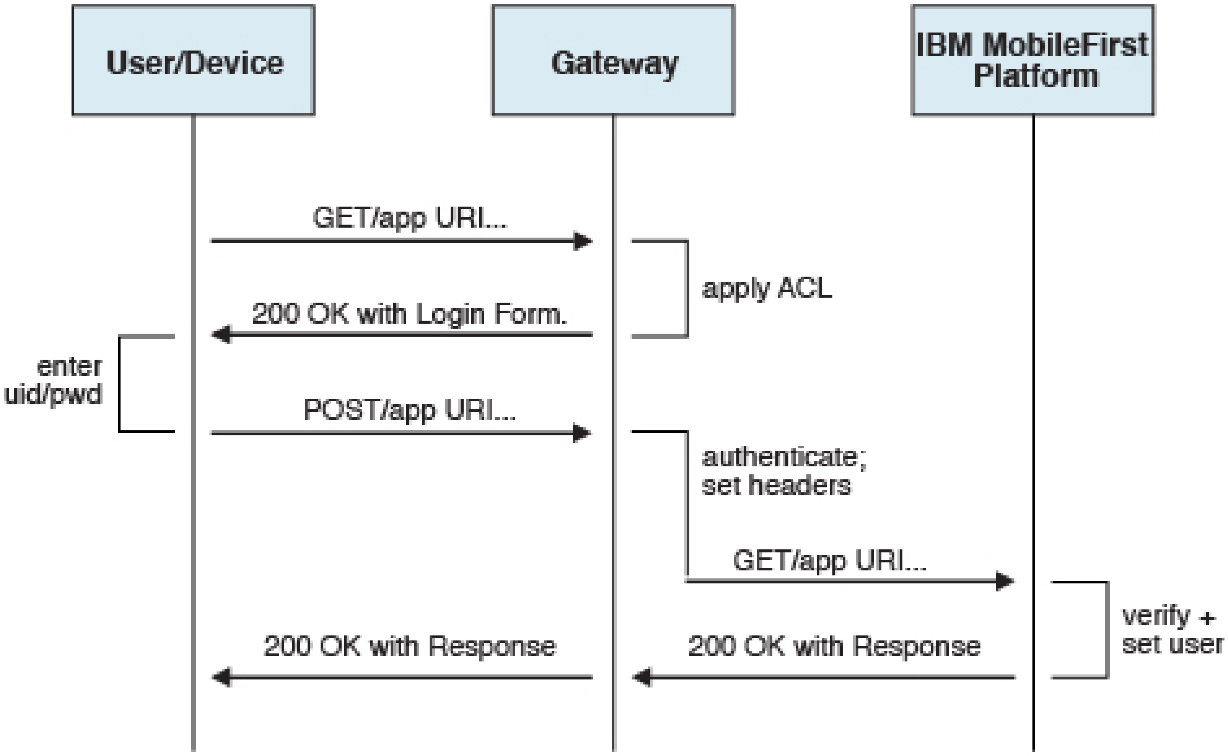


Figure 14-5. Authentication flow

This configuration was tested with DataPower and IBM Security Access Manager for header-based authentication and LTPA-based authentication.

Header-based authentication

- On successful authentication, the gateway forwards a custom HTTP header with the user name or ID to IBM MobileFirst Platform Foundation for iOS.
- IBM MobileFirst Platform Foundation for iOS is configured to use `HeaderAuthenticator` and `HeaderLoginModule` on either Tomcat or WebSphere Application Server.

LTPA-based authentication

- On successful authentication, the gateway forwards an LTPA token (in the form of an HTTP cookie) to IBM MobileFirst Platform Foundation for iOS, which validates the LTPA token and creates a caller principal.
- IBM MobileFirst Platform Foundation for iOS on WebSphere Application Server is configured to use the WebSphereFormBasedAuthenticator and WebSphereLoginModule classes. See “LTPA authenticator” on page 8-290 and “WASLTPAModule login module” on page 8-265.

For more information about using lightweight third-party authentication (LTPA) tokens with IBM MobileFirst Platform Foundation for iOS, see “MobileFirst Security and LTPA” on page 11-91.

For information about using WebSphere DataPower as a reverse proxy, see “Integrating with DataPower as a security gateway and reverse proxy” on page 6-157.

Related information

- “Integration with IBM WebSphere DataPower as a security gateway and reverse proxy” on page 6-156
- “Using a reverse proxy with server farm and WebSphere Application Server Network Deployment topologies” on page 6-18
- “Reverse proxy with LTPA” on page 11-97
- “Typical topologies of a MobileFirst instance in an extranet infrastructure” on page 6-299

Integration with IBM Endpoint Manager

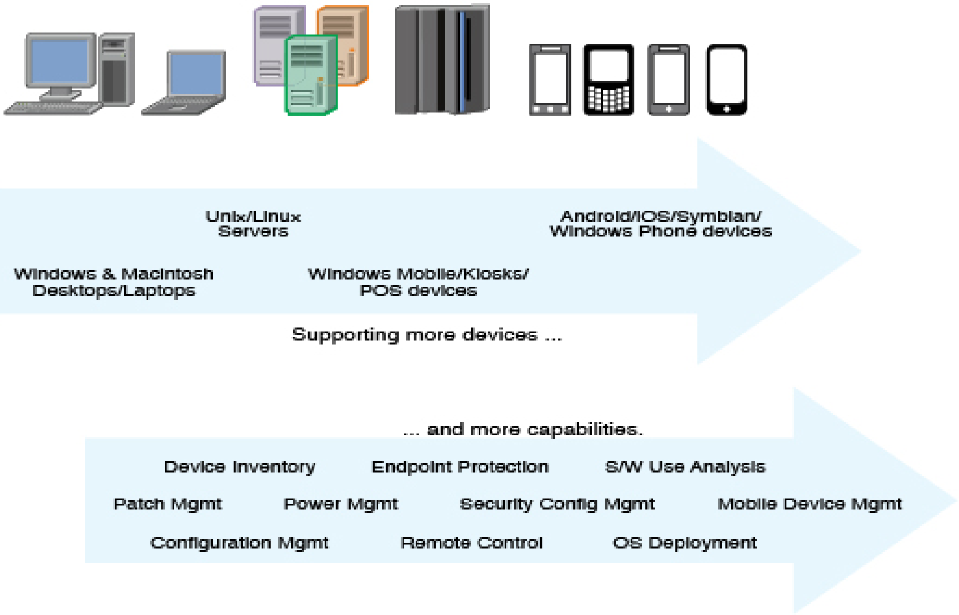
In a MobileFirst environment, you can implement an IBM Endpoint Manager architecture to make your enterprise devices and applications benefit from endpoint management features, such as data security, compliance, and unified infrastructure.

IBM Endpoint Manager for Mobile Devices

The features and architecture of IBM Endpoint Manager for Mobile Devices are described here.

Features

With IBM MobileFirst Platform Foundation for iOS, you can integrate the security features that IBM Endpoint Manager provides. The purpose of IBM Endpoint Manager is to deliver a unified solution for the management of systems and security, for all enterprise devices.



IBM Endpoint Manager for Mobile Devices provides security capabilities in the following areas:

- Enterprise Access Management: Configuration of email, VPN, and Wi-Fi.
- 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 and manageable container for BYOD (Bring Your Own Device) devices. The Divide app provides a workspace that mimics device capabilities. Because this workspace is isolated from the rest of the device, Divide can manage information separately and safely.

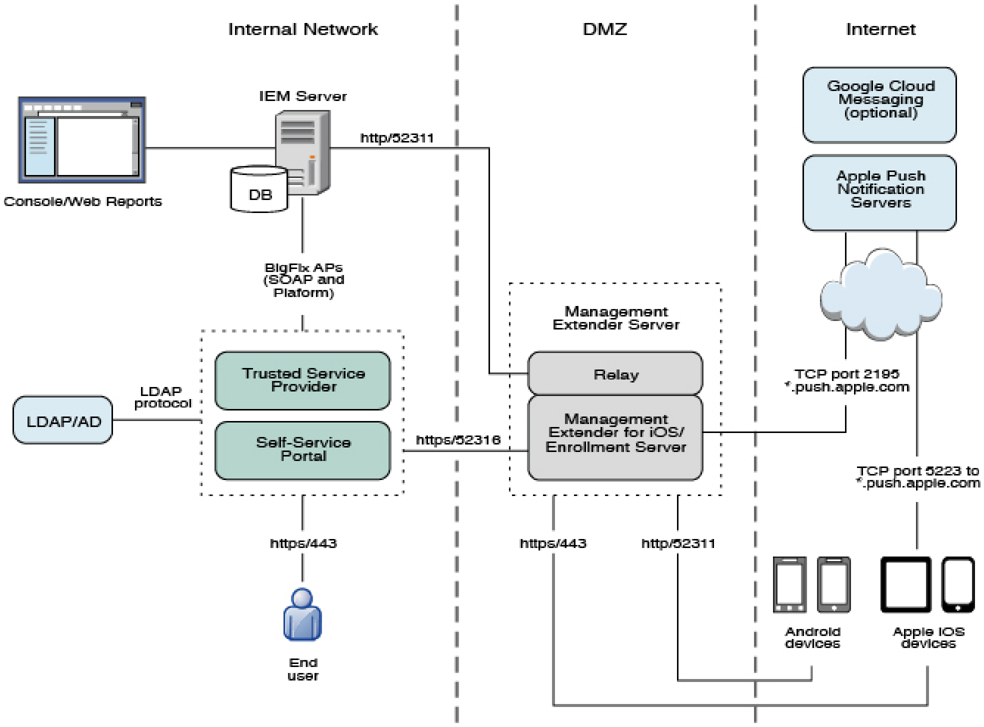
Architecture

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 iOS devices through the IBM Mobile Client. This approach provides the full set of capabilities through the usage of Apple's MDM APIs and the Push Notification Server infrastructure (APNS).
- An email-based management through Exchange (Active Sync) and Lotus® Traveler (IBM Sync). In this approach, iOS is 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 Wi-Fi 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 "Features" on page 14-5.

- PIM is available through NitroDesk TouchDown.

The following diagram shows an architectural overview of a production-level, agent-based, MDM API-based implementation with IBM Endpoint Manager for Mobile Devices.


Internal Network
DMZ
Internet
IEM Server
Console/Web Reports
DB
http/52311
BigFix APs (SOAP and Platform)
Google Cloud Messaging (optional)
Apple Push Notification Servers
Management Extender Server
Relay
Management Extender for iOS/ Enrollment Server
LDAP protocol
LDAP/AD
Trusted Service Provider
Self-Service Portal
https/52316
TCP port 2195 *.push.apple.com
TCP port 5223 to *.push.apple.com
https/443
https/443
http/52311
Android devices
Apple IOS devices
End user

End-point management with IBM Endpoint Manager

IBM Endpoint Manager for Mobile Devices provides features to manage devices in a MobileFirst environment.

IBM MobileFirst Platform Foundation for iOS 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 MobileFirst Platform Foundation for iOS and IBM Endpoint Manager for Mobile Devices, as shown in Figure 14-6 on page 14-8.

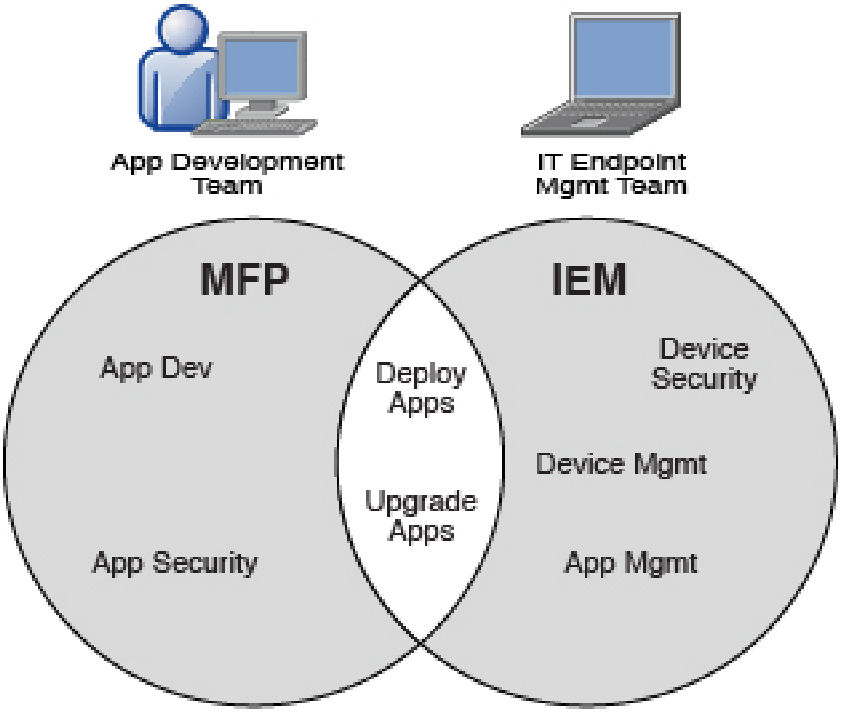


Figure 14-6. IBM MobileFirst Platform Foundation for iOS and IBM Endpoint Manager management capabilities

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 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 your enterprise devices; that is, smartphones, tablets, desktops, notebooks, and servers.

Integration with IBM Tealeaf

The use of IBM Tealeaf is described here.

IBM Tealeaf CX Mobile helps customers apply the power of Tealeaf powerful solutions for customer experience management to their mobile websites, and native 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 is provided as a set of libraries. To use it, you must take steps on both your client and your server.

For more information, see the Product Integration pages.

IBM Tealeaf client-side integration

To make the IBM Tealeaf client SDK available for development, follow the iOS specific instructions for IBM Tealeaf to place the libraries and configuration files in the appropriate location of your MobileFirst project file system for each supported environment.

Procedure

Manually edit the `application-descriptor.xml` file to manage this optional feature.

Results

After the libraries and configuration files are in the correct directory, the IBM Tealeaf client-side API is on your Java class path and available in XCode (Objective-C), and the TLT global variable is available in your JavaScript code.

Important:

Be aware of the following effects:

- 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.

Therefore, 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 edit the `pList` file and how to use the IBM Tealeaf CX Mobile API, see the Tealeaf documentation indicated in the Tealeaf CX Mobile page of the Developer Center website for IBM MobileFirst Platform.

IBM Tealeaf server-side integration

To aggregate the data that is collected in client applications by the IBM Tealeaf client SDK, you must install the IBM Tealeaf Mobile CX server.

For more information about installation, configuration, and usage, see Tealeaf CX Mobile.

Integration with IBM Trusteer

You can use IBM Trusteer to collect mobile device risk factors. By providing these to your mobile app, you can restrict mobile app functionality by risk levels.

IBM MobileFirst Platform Foundation for iOS supports the following versions of IBM Trusteer Mobile SDK:

- IBM Trusteer Mobile SDK Version 3.6 and later fix packs
- IBM Trusteer Mobile SDK Version 4.0 and later fix packs

IBM MobileFirst Platform Foundation for iOS supports full integration with the IBM Trusteer Mobile SDK for iOS applications.

After the Trusteer Mobile SDK is installed, you must configure MobileFirst Server to support it:

- “MobileFirst security overview” on page 11-87 and “MobileFirst security configuration” on page 11-89 provide an overview of general MobileFirst security configuration.
- “Configuring the MobileFirst Server for Trusteer” on page 11-97 provides steps for configuring MobileFirst Server to support Trusteer.

For more information, see Trusteer for iOS in the Development Center.

Integrating IBM Trusteer for iOS

You might want to integrate Trusteer with IBM MobileFirst Platform Foundation for iOS.

Before you begin

- If you are using a Trusteer compressed archive:
  1. Make sure that you have the following files:
     - Trusteer Mobile iOS library: `libtas_full.a`.
     - A MobileFirst-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.
  2. In your file system, create a `tas` folder and place in it the files that are listed previously.
  3. Continue from step 1.

Note: Starting with its version 4.0, Trusteer supports the arm64 architecture. Earlier versions of Trusteer do not.

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.
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 **Build Settings** > **Deployment**, set **Strip Linked Product** to **NO**.
10. In the Xcode **Project Navigator**, drag `tas.license` into the **Resources** group.
11. In the dialog box that opens, click **Finish**.
12. Open the `tas.license` file and check that the values for `vendorId`, `clientId`, and `clientKey` match the licensing information that was provided by Trusteer.
13. Initialize the Trusteer Mobile SDK. By default, the Trusteer Mobile SDK is initialized automatically when you integrate the SDK into a MobileFirst project. However, to provide more flexibility and eliminate potential initialization failure, it is recommended that you disable the automatic SDK initialization and instead initialize the SDK manually:
    a. Disable the automatic initialization of the Trusteer Mobile SDK by setting the **`TRUSTEER_AUTO_INIT`** property in your client properties file (`worklight.plist`) to `false`:

    ```
    <key>TRUSTEER_AUTO_INIT</key>
    <false/>
    ```

    b. Manually initialize the SDK. You can do this either by using the Trusteer Mobile SDK according to the Trusteer documentation, or by using the MobileFirst Trusteer API:
    - In your native iOS code, call the init method of the WLTrusteer sharedInstance class method:

      ```
      [[WLTrusteer sharedInstance] init];
      ```

Integration with IBM WebSphere DataPower

Use IBM WebSphere DataPower as a reverse proxy and security gateway for handling inbound mobile traffic, or as a proxy and gateway for handling outbound traffic to a notification mediator.

IBM WebSphere DataPower (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 and authenticating mobile clients. For information about configuring DataPower as a reverse proxy and security gateway between MobileFirst applications and MobileFirst Server, see “Integration with IBM WebSphere DataPower as a security gateway and reverse proxy” on page 6-156.

DataPower can also be used as a gateway for monitoring and routing outbound connections when required by corporate policy. For information about configuring DataPower as a proxy between MobileFirst Server and a notification mediator (Apple Push Notification Service) for pushing notifications to mobile applications, see “Integrating with WebSphere DataPower as a push notification proxy” on page 14-12..

Related information

- "Integration with IBM WebSphere DataPower as a security gateway and reverse proxy" on page 6-156
- "Integrating IBM WebSphere DataPower with a cluster of instances of MobileFirst Server" on page 6-320

Integrating with WebSphere DataPower as a push notification proxy

Use IBM WebSphere DataPower as a gateway and proxy for pushing outbound notifications from MobileFirst Server to a notification mediator.

About this task

IBM MobileFirst Platform Foundation for iOS makes outbound connections to a notification mediator, APNS (Apple Push Notification Service), to push notifications to mobile applications. You can configure IBM WebSphere DataPower (DataPower) as a proxy between MobileFirst Server and your notification mediator.

Procedure

- You must configure both DataPower and MobileFirst Server, regardless of your notification mediator.
- For more information and detailed step-by-step instructions, see the developerWorks article Using WebSphere DataPower as a push notification proxy for MobileFirst mobile applications.

Related information

Integration with IBM WebSphere DataPower as a security gateway and reverse proxy
Protect your mobile-application traffic by using IBM WebSphere DataPower as a reverse proxy and security gateway in the DMZ between client applications and MobileFirst Server.

Integrating IBM WebSphere DataPower with a cluster of instances of MobileFirst Server
You can use IBM WebSphere DataPower as a gateway for all incoming connections for IBM MobileFirst Platform Foundation for iOS and Application Center, and IBM HTTP Server (IHS) for load-balancing MobileFirst Server that are deployed on an IBM WebSphere Application Server 8.5 cluster or a Liberty profile server farm.

More about integration

More 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

Reference

Reference information about Ant tasks, configuration sample files

CLI Commands

You can use IBM MobileFirst Platform Command Line Interface for iOS to create apps from the command line.

adapter add

Creates a new adapter, which is generated in the **adapters** folder of the current project.

Syntax

```
mfp adapter add [<name> --type|-t <type> [--jsonstore|-j] [--ussd|-u]
[--package|-p <package>]]
```

Parameters

<name>
The name of the generated adapter.

<type>
The type of adapter. Valid values: `castiron`, `http`, `java`, `jco`, `jms`, `sap`, and `sql`.

[--jsonstore|-j]
Your choice of JSONStore procedures. JSONStore procedures are not valid for JCo and Java adapters.

[--ussd|-u]
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

[--package|-p <package>]
The name of the package that contains the Java adapter classes. Package option is valid only for Java adapters.

adapter call

Calls an adapter procedure on the MobileFirst Server.

Syntax

Choose one of the following syntaxes.

```
mfp adapter call [<adapter>/<procedure> [--type|-t get|post] [--file|-f
<args.json>]]
```

```
mfp adapter call [<adapter>/<procedure>

    [--type|-t get|post]

    [--form|-fp <form_parameters>]

    [--header|-h <header_parameters>]
```

[--query|-q <query_parameters>]

]

Parameters

[--file|-f]
The JSON file that contains the required arguments.

[--type|-t]
Indicates the request type. Valid options: **get** or **post**. If you are using Java adapters, the type also includes **put** and **delete**.

Note: If the **type** parameter is not defined, the `adapter call` command uses **get** by default. Type cannot be passed in a JSON file.

[--form|-fp]
The form parameters to pass with the procedure. For Java Adapters and Direct Mode use only. Example: `--form "username:user"`.

[--header|-h]
The header parameters to pass with the procedure. For Java Adapters and Direct Mode use only. Example: `--header "password:pass"`.

[--query|-q]
The query parameters to pass with the procedure. For Java Adapters and Direct Mode use only. Example: `--query "date:01/01/2015"`.

Usage

You can use the adapter call command in interactive mode or in direct mode.

Note: In direct mode, any `string` parameters, such as `"stringParameter"` in these examples, must have their quotation marks escaped, even if the string object is within an Array or Object object. Escaping `string` parameters is not necessary in interactive mode.

- Interactive: **`$ mfp adapter call`**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **`$ mfp adapter call adapterName/procedure`**.
- Direct with multiple parameters: **`$ mfp adapter call adapterName/procedure \"stringParameter\", 2, true`**.
- Direct with an array: **`$ mfp adapter call adapterName/procedure [\"stringParameter\", 2, true]`**
- Direct with a JSON file parameter: **`$ mfp adapter call adapterName/procedure --file ./myArgs.json`**

  JSON file

  ```
  [
    "world"
  ]
  ```

  JSON file for Java adapter

  ```
  {
    "FORM":{"username":"user","password":"pass"},
    "QUERY":{"index":6},
    "HEADER":{"Date":"01/01/2015"}
  }
  ```
- Direct with POST method: **`$ mfp adapter call adapterName/procedure/{path_parameter} -fp "username:user1" --type post`**. To use the **path**

parameters, add them to the `adapter/procedure` path. Replace the {**`path_parameter`**} with your value. For example:

  - Before: **`myAdapter/users/{username}`**
  - After: **`myAdapter/users/StanLee`**

- Direct with Java adapter: **`$ mfp adapter call "adapterName/procedure/{pathParameter with space}" --form "username:user" --header "Date:01/01/2015" --query "index:6" --type post`**

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

add adapter

Deprecated in V7.0.0. Use **`adapter add`** instead. Creates a new adapter, which is generated into the **adapters** folder of your project.

Syntax

```
mfp add adapter [<name> --type|-t <adaptertype> [--jsonstore|-j]
[--ussd|-u] [--package|-p]]
```

Parameters

`<name>`
The name that you want for the generated adapter.

`<adaptertype>`
The type of adapter. Valid values: `http`, `castiron`, `sap`, `jco`, and `java`.

`[--jsonstore|-j]`
Your choice of JSONStore procedures. JSONStore procedures are not valid for JCo and Java adapters.

`[--ussd|-u]`
The option for the Unstructured Supplementary Service Data (USSD) communication technology.

`[--package|-p]`
The name of the package that contains the Java adapter classes. Package option is valid only for Java adapters.

add api

Generates a new native API into the apps folder of the current project.

Syntax

```
mfp add api [<name>]
```

Parameters

`<name>`
The name that you want for the generated native API.

Usage

Run this command in your current working directory.

add feature

Adds optional features to your app.

Syntax

```
mfp add feature [fips|jsonstore|tealeaf]
```

Parameters

fips
FIPS 140-2

jsonstore
JSONStore

tealeaf
IBM Tealeaf SDK

(deprecated) bd

Builds and deploys to the local test server the set of MobileFirst resources that are most local to the current working directory.

mfp bd is deprecated, see Table 3-1 on page 3-14. Use **mfp push** instead.

Syntax

```
mfp bd
```

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

(deprecated) build

Builds the set of MobileFirst resources that are most local to the current working directory.

mfp build is deprecated, see Table 3-1 on page 3-14. Use **mfp push** instead.

Syntax

```
mfp build
```

Usage

- If you run the command from the root folder of your 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.

config

Configure settings of your project.

You can configure global settings, that apply to all projects, and client app settings that are specific to individual projects. You can use the **config** command in interactive mode or direct mode. Using the **config** modifies settings that are defined in the `application-descriptor.xml` file, the `.mfp_cli_config.json` file (located in the same directory as the `application-descriptor.xml` file), and the `$HOME/.ibm/mobilefirst/config_cli.json` file. Using the **config** command modifies the values in these files.

Syntax

mfp config [<setting>[<value>]]

Parameters

<setting>
Name of the setting. For example: `app_author_email` is used to specify the email address of the application author.

<value>
Value to which you are changing the setting. For example: For the `app_author_email` setting, the value might be `"myname@company.com"`.

Usage

Interactive mode

```
$ mfp config
```

In interactive mode you have two options: **Global** or **Client App**. You are prompted for the options that you want to specify.

Global
You can configure the following global settings for your projects.

development_browser
The web browser that you use to preview your app.

default_preview_type
Set the default preview type of your app. You can choose from either the Browser or the Mobile Browser Simulator.

global_app_id_default_prefix
Value with which you can prefix all of your apps. Changing this will change the **application id** attribute of the `application-descriptor.xml` file.

Client App
You can configure settings, or *keys*, that apply to only a specific project. You must be in a MobileFirst project directory of the project that you want to modify when you change these settings.

Common Keys
Configure keys common to all apps.

Platform **Keys**
Keys that are specific to your target platform.

Direct mode
In direct mode you have two options: **Direct mode setting** or **Direct mode setting and value**:

Direct mode setting
Displays the current value of the entered setting.

Syntax

```
$ mfp config [<setting>]
```

Example

```
$ mfp config app_author_email
Current value for app_author_email: myname@company.com
```

Direct mode setting and value

Specifies the setting to change and its value.

Syntax

```
$ mfp config [<setting> [<value>] ]
```

Example

```
$ mfp config app_author_email myname@company.com
```

console

Opens the MobileFirst Operations Console in your default browser for the default or specified server.

Syntax

mfp console [*server*]

Parameters

server

where *server* is the name of the MobileFirst server definition. If *server* is not specified, then the default server name is used.

Usage

You can run the **mfp console** command from any directory.

create

Creates a new project in the current working directory. This project contains an iOS app in the apps folder.

Syntax

mfp create [<name> --app|-a <app>]

Parameters

<name>
The name of the project that you are creating.

--app|-a <app>
The name of the app that you are creating.

Usage

- To create a project named MyProject: **$ mfp create MyProject --app MyApp**

(deprecated) create-server

This command does not need to be run manually because the built-in development server is automatically created and started as needed. Creates a new WebSphere Application Server Liberty application server in your default folder.

mfp create-server is deprecated, see Table 3-1 on page 3-14. Use **mfp server create** instead.

Syntax

```
mfp create-server
```

Usage

```
$ mfp create-server
```

(deprecated) deploy

Deploys to the local test server the set of resources that are found in the current working directory.

mfp deploy is deprecated, see Table 3-1 on page 3-14. Use **mfp push** instead.

Syntax

```
mfp deploy
```

Usage

- If you run the command from the root folder of a 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.

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

help

Displays either information about the general use of the CLI, or detailed help for each command.

Syntax

```
mfp help [<command>]
```

Usage

- Information about the use of the CLI: **`$ mfp help`**.
- Information about the **invoke** command: **`$ mfp help invoke`**.
- Information about the IBM MobileFirst Platform Foundation for iOS development environment: **`$ mfp help intro`**

info

Generates information about your environment. This information contains the operating system, the memory consumption on your machine, the node version, and the CLI version.

Syntax

```
mfp info
```

Usage

If your current directory is a project folder, it also displays information about your project.

invoke

This command is deprecated. You can use the **adapter call** command instead. The **invoke** command starts a procedure for a specified adapter on the MobileFirst Server.

Syntax

```
mfp invoke [<adapter>:<procedure>[\"<json array>\"|--file|-f <file path>]]
```

Parameters

<adapter>
The name of the adapter you are invoking.

<procedure>
The adapter's procedure that you are invoking.

<json array>
JSON array you are passing to the adapter's procedure.

--file|-f <file path>
File path to the JSON array file.

Usage

Pass the arguments as a comma-separated list. If you enter the command with no arguments, you are prompted for an adapter and procedure, based on what is available in the current working directory of your project.

- Interactive: **$ mfp invoke**, then follow the prompts by using the arrow keys.
- Direct with no parameters: **$ mfp invoke adapterName:function**.
- Direct with a JSON array parameter: **$ mfp invoke adapterName:function [\"string\", 2, true]**.
- Direct with a JSON file parameter: **$ mfp invoke adapterName:function --file ./myArts.json**.

logs

Displays the path to the local test server logs.

Syntax

```
mfp logs
```

Usage

You can run this command from any directory: **$ mfp logs**

push

This command replaces the previous **mfp build**, **mfp deploy**, and **mfp bd** commands. Prepares and transfers the relevant assets to either a local or remote MobileFirst Server.

Syntax

```
mfp push [<server>] [<runtime>] [--password|-p <password>] [--minify|-m]
[--concatenate|-c] [--nosend|-n]
```

Parameters

<server>
Name of predefined server to push to. This parameter is optional, and the default server is used if you do not provide this parameter. For a list of defined servers, run **mfp server info**. Define new servers by running **mfp server add**.

<runtime>

Runtime of your back-end project running on the server.

- If this value is not specified and the **mfp push** command was previously run for this application, the command uses the runtime value from the last time **mfp push**was run.
- If this value is not specified and this is the first time **mfp push** is run for this application, the runtime value is obtained from the server:
  - if there is a single runtime on the server, it is used.
  - if there are multiple runtimes, you are prompted to select a runtime.
  - if there are no runtimes on the server, the command exits with an error.
- If the **mfp push** command is run from within a MobileFirst project folder, the runtime value must exactly match the project runtime name; otherwise the command exits with an error.

--password|-p <password>
The password for the admin login ID. If you do not supply this parameter and it is not in server configuration, you are prompted for the admin password on all server management tasks.

--minify|-m
Enables minification of web assets for hybrid apps. By default, no minification is performed during a hybrid app push.

--concatenate|-c
Enables concatenation of web assets for hybrid apps. By default, no concatenation is performed during a hybrid app push.

--nosend|-n

Prevents the deployment of artifacts to the target server. You can use this in automated build environments where the build and deployment steps are isolated.

The following actions are taken:

- MobileFirst Project:

  - WAR and `.wladapter` are created.
  - If the target server is a local server and if the WAR file was not deployed to this server before: the WAR file is built and deployed to the server (no matter where you run the command from).
  - If the target server is a local server and if the WAR file was deployed to this server before: the WAR is not deployed again, unless you run the **`mfp push`** command from the root directory of your project (in which case the WAR is redeployed).
- MobileFirst Native app: `application-descriptor.xml` and `.wlapp` are created and updated.
- iOS native app: `worklight.plist`, `application-descriptor.xml` and `.wlapp` are created and updated.

Usage

Run this command in the current working directory, which is a child of a project.

```
$ mfp push
```

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

restart

Restarts the local test server.

Syntax

```
mfp restart
```

Usage

Run this command in the current working directory, which is a child of a project. On completion, a message indicates a successful start, and control is returned to the command line.

```
$ mfp restart
```

run

Starts the local test server.

Syntax

```
mfp run
```

Usage

Run this command in the current working directory, which is a child of a project. This command does not return control to the command line. It displays new status messages from the server until the server is stopped.

```
$ mfp run
```

server add

Adds a server definition.

Syntax

```
mfp server add
    [<name> --url|-u <url> [--login|-l <login>]
    [--password|-p <password>]
    [--contextroot|-c <context root>]
    [--setdefault|-s]
    ]
```

Parameters

<name>
The name of the server to use as an alias.

--url|-u <url>
The fully qualified URL of the server. The syntax must include the full: `protocol://name.domain:port`. The IP address might also be used instead of `name.domain`. This value is required if the server name is supplied on the command line.

--login|-l <login>
The admin login ID that is used to manage the MobileFirst Server. If not set, defaults to `admin`.

--password|-p <password>
The password for the admin login ID. If you do not set this parameter, you are prompted for the admin password on all server management tasks. There is no default password.

--contextroot|-c <context root>
The context root of the MobileFirst administration services. If not set, defaults to `worklightadmin`.

--setdefault|-s
Providing this flag makes this new server the new default server profile. You can switch the default by using the **mfp server edit** command.

Usage

- Interactive mode: **$ mfp server add**
- Direct mode: **$ server add name --url http://acme.com:10080 --setdefault**

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see “Specifying a trusted SSL certificate” on page 6-184.

server create

Create or recreate a local development server. A server profile is automatically generated for this server. This command replaces the deprecated **mfp create-server** command.

Syntax

`mfp server create`

Usage

`$ mfp server create`.

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

server edit

Edits an existing server definition.

Syntax

```
mfp server edit
    [<name> [--name|-n <newName>] [--url|-u <newUrl>]
    [--login|-l <newLogin>] [--password|-p <newPassword>]
    [--contextroot|-c <newContextRoot>] [--setdefault|-s]]
```

Parameters

`<name>`
The name of the server to modify.

`--name|-n <newName>`
The new name for this alias. Renames this entry.

`--url|-u <newUrl>`
The fully qualified URL of the server. The syntax must include the full: `protocol://name.domain:port`. The IP address can also be used instead of `name.domain`. This value is required if the server name is supplied on the command line.

`--login|-l <newLogin>`
The admin login ID used to manage the MobileFirst Server.

`--password|-p <newPassword>`
The password for the admin login ID. If not supplied, you are prompted for the admin password on all server management tasks.

`--contextroot|-c <newContextRoot>`
The context root of the MobileFirst administration services.

`--setdefault|-s`
Providing this flag makes this new server the new default server profile.

Usage

- Interactive mode: **`$ mfp server edit`**.
- Direct mode: **`$ mfp server edit staging --name preprod --setdefault`**

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve` `runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

server info

Provides information for all servers or a specific server.

Syntax

```
mfp server info [<name> [--password|-p <password>]]
```

Parameters

<name>
The name of a specific server that you want information about.

--password|-p <password>
The password for the admin login ID. If not supplied, you are prompted for the admin password on all server management tasks. Not applicable if you don't provide a server.

Usage

There is no interactive mode for this command, other than the possible prompt for a server's password.

- **`$ mfp server info`**
- **`$ mfp server info name`**

Important: If the certificate of the server is not recognized as a trusted certificate, the following error is displayed when you run this command: `Failed to retrieve runtime information: CERT_UNTRUSTED`. To add your certificate to the list of trusted certificates, see "Specifying a trusted SSL certificate" on page 6-184.

server remove

Removes an existing server definition.

Syntax

```
mfp server remove [<name>]
```

Parameters

<name>
The name of the server to remove.

Usage

- Interactive mode: **`$ mfp server mode`**.
- Direct mode: **`$ mfp server remove name`**

start

Starts the local test server.

Syntax

```
mfp start
```

Usage

Run this command in the current working directory, which is a child of a project. On completion, a message indicates that a successful start and control is returned to the command line.

```
$ mfp start
```

status

Displays the status of the local test server, which is either running or stopped.

Syntax

```
mfp status
```

Usage

Run the command in the current working directory, which is a child of a project.

```
$ mfp status
```

stop

Stops the local test server.

Syntax

```
mfp stop [--kill|-k]
```

Parameters

--kill|-k
If you are unable to stop your server, this flag causes your operating system to send signals to the running server process to request the process to end.

Usage

Run this command in the current working directory, which is a child of a project.

```
$ mfp stop
```

```
$ mfp stop --kill
```

Ant configuredatabase task reference

Reference information for the **configuredatabase** Ant task. This reference information is for relational databases only. It does not apply to Cloudant.

Overview

Note: In MobileFirst Server V7.1.0, the reports database (WLREPORT) is deprecated. Use "Operational analytics" on page 13-6 instead. Note that setting up the reports database is optional in this release and earlier releases.

The **configuredatabase** Ant task creates the relational databases that are used by MobileFirst Administration Services, the MobileFirst runtime, and the Application

Center Services. This Ant task configures a relational database for a MobileFirst project, or the Application Center Services, through the following actions:

- Checks whether the MobileFirst tables exist and creates them if necessary.
- If the tables exist for an older version of IBM MobileFirst Platform Foundation for iOS, migrates them to the current version.
- If the tables exist for the current version of IBM MobileFirst Platform Foundation for iOS, does nothing.

In addition, if one of the following conditions is met:

- The DBMS type is Derby.
- An inner element `<dba>` is present.
- The DBMS type is DB2, and the specified user has the permissions to create databases.

Then, the task can have the following effects:

- Create the database if necessary (except for DB2, Oracle 12c, and Cloudant).
- Create a user, if necessary, and grants that user access rights to the database.

Note: The **`configuredatabase`** Ant task has not effect if you use it with Cloudant.

In IBM Worklight Foundation V6.2.0, a new database was introduced, which is referenced with kind **`WorklightAdmin`** for Administration Services. This database can support one MobileFirst runtime or more, and can handle the artifacts of those MobileFirst runtimes.

Important: If you upgrade from an IBM Worklight version earlier that V6.2.0, you must also migrate the data from the IBM Worklight runtime to the new database for MobileFirst Administration Services. IBM Worklight Foundation V6.2.0 introduced 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 15-1. Attributes for the **`configuredatabase`** *Ant task*

Attribute	Description	Required	Default
`kind`	Type of database: In MobileFirst Server: **`Worklight`**, **`WorklightReports`**, or **`WorklightAdmin`** In Application Center: **`ApplicationCenter`**	Yes	None

IBM MobileFirst Platform Foundation for iOS V7.1.0 supports four kinds of database: MobileFirst runtimes use **`Worklight`** and **`WorklightReports`** databases. MobileFirst Administration Services use the **`WorklightAdmin`** database. Application Center uses the **`ApplicationCenter`** database.

The `configuredatabase` task supports the following elements:

Table 15-2. Inner elements for the <`configuredatabase`> Ant task

Element	Description	Count
`derby`	Parameters for Derby	0..1

Table 15-2. Inner elements for the `<configuredatabase>` Ant task (continued)

Element	Description	Count
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
admindatabase	Parameters for migrating data from IBM Worklight V6.1.x runtime to IBM MobileFirst Platform Foundation for iOS V7.1.0 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 15-3. 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 MobileFirst runtime database to the MobileFirst Administration Services database. This element is mandatory when you migrate your IBM Worklight runtime projects from V6.1.x and the `kind` attribute of **configuredatabase** is **Worklight**.

The `admindatabase` element has the following attribute.

Table 15-4. Attribute for the <admindatabase> element

Attribute	Description	Required	Default
runtimeContextRoot	Context root of the MobileFirst runtime	Yes	None

Because the MobileFirst Administration Services can handle one or more MobileFirst runtimes, you must reference a specific context root for each runtime. Use the `runtimeContextRoot` attribute to specify this context root. After MobileFirst data is migrated, you cannot change the context root of the MobileFirst runtime, unless the MobileFirst Administration Services database is removed and a new database is created.

The `<admindatabase>` element supports the following elements.

Table 15-5. 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

Table 15-5. Inner elements for the <admindatabase> element (continued)

Element	Description	Count
oracle	Parameters for Oracle	0..1

Apache Derby

The <derby> element has the following attributes:

Table 15-6. Attributes for the <derby> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT*, *WLREPORT*, *WLADMIN*, or *APPCNTR*, depending on kind.
datadir	Directory that contains the databases	Yes	None
schema	Schema name	No	*WORKLIGHT*, *WORKLIGHT*, *WLADMINISTRATOR*, or *APPCENTER*, depending on kind

The <derby> element supports the following elements:

Table 15-7. 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 15-8. Attributes for the <db2> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT*, *WLREPORT*, *WLADMIN*, or *APPCNTR*, 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 15-9. Inner elements for the <db2> element

Element	Description	Count
property	JDBC connection property	0..∞
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 15-10. 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 11-20.

MySQL

The element <mysql> has the following attributes:

Table 15-11. Attributes for the <mysql> element

Attribute	Description	Required	Default
database	Database name	No	*WRKLGHT*, *WLREPORT*, *WLADMIN*, or *APPCNTR*, 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 15-12. 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 15-13. 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 15-14. 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 15-15. 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 MobileFirst Server version. You do not have to specify the inner elements `<dba>` or `<client>`.

Oracle

The element `<oracle>` has the following attributes:

Table 15-16. Attributes for the <oracle> element

Attribute	Description	Required	Default
database	Database name, or Oracle service name **Note:** You must always use a service name to connect to a PDB database.	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. See the note under this table.	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

Note: For the **user** attribute, use preferably a user name in uppercase letters. Oracle user names are generally in uppercase letters. Unlike other database tools, the **configuredatabase** Ant task does not convert lowercase letters to uppercase letters in the user name. If the **configuredatabase** Ant task fails to connect to your database, try to enter the value for the **user** attribute in uppercase letters.

For more information about Oracle user accounts, see Overview of Authentication Methods.

The `<oracle>` element supports the following elements:

Table 15-17. 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 15-18. 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 11-32.

Alternatively, you can use the `<oracle>` element with the following attributes:

Table 15-19. 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 MobileFirst Server version. You do not have to specify the inner element `<dba>`.

Customizing the database connection with JDBC properties

You can customize the database connection with JDBC properties. This can be used to define the security (SSL) of the connection to the database server, timeouts, or JDBC traces.

About this task

To customize the database connection, you must add `<property>` elements to the database elements for the tasks **configuredatabase**, **configureapplicationserver**, and **installworklightadmin**. The JDBC properties are used by the Ant tasks when connecting to the database, and by the application server data source installed by **configureapplicationserver** and **installworklightadmin**.

You can find in the following procedure an example that defines the properties to set the command timeout for DB2 for the connection to the administration database.

Note: The steps for customization that are described in this page do not apply if you are using a Cloudant database. For customizing Cloudant, see Property keys and values for Cloudant configuration and Table 6-29 on page 6-110.

Procedure

1. From the "Sample configuration files" on page 15-66, select the file `configure-liberty-db2.xml`, and copy it to your working directory.
2. Review the Properties for the IBM Data Server Driver for JDBC and SQLJ in the DB2 for Linux UNIX and Windows user documentation.
3. Edit the Ant file to add the relevant JDBC properties in **`configuredatabase`**, **`configureapplicationserver`**, and **`intallworklightadmin`**.

```
<target name="admdatabases">
  <configuredatabase kind="WorklightAdmin">
    <db2 database="${database.db2.wladmin.dbname}"
         server="${database.db2.host}"
         instance="${database.db2.instance}"
         user="${database.db2.wladmin.username}"
         port= "${database.db2.port}"
         schema = "${database.db2.wladmin.schema}"
         password="${database.db2.wladmin.password}">

      <property name="commandTimeout" value="10"/>

    </db2>

    (...)

<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.initialpassword}"/>
    <database kind="WorklightAdmin">
      <db2 database="${database.db2.wladmin.dbname}"
           server="${database.db2.host}"
           user="${database.db2.wladmin.username}"
           port= "${database.db2.port}"
           schema = "${database.db2.wladmin.schema}"
           password="${database.db2.wladmin.password}">

        <property name="commandTimeout" value="10"/>

      </db2>

      (...)
```

Encrypting database password with Ant tasks for Liberty

You can use Ant tasks to encrypt database passwords for the WebSphere Application Server Liberty server with the `aes` algorithm.

About this task

By default, Ant tasks encrypt passwords for the WebSphere Application Server Liberty server with the `xor` algorithm. You can encrypt them with the `aes` algorithm, but only with the default key.

You can use passwords that are already encrypted as input for the Ant tasks, for example if the Ant tasks are used by persons that should not have access to database production passwords.

Procedure

1. From the "Sample configuration files" on page 15-66 select the file `configure-liberty-<database>.xml`, and copy it to your working directory.
2. Add a **`libertyEncoding`** attribute with the value `none` in the `websphereapplicationserver` element of the **`configureApplicationServer`** and **`installWorklightAdmin`** Ant tasks. For more information, see table 6 of "Ant tasks for installation of MobileFirst runtime environments" on page 15-32.
3. For each database element of **`configureApplicationServer`** and **`installWorklightAdmin`**, add an attribute **`validate`** with the value `false`.

```
<target name="adminstall">
  <installworklightadmin >
    [...]
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
                                  profile="${appserver.was.profile}"
                                  libertyEncoding="none">
        <server name="${appserver.was85liberty.serverInstance}"/>
      </websphereapplicationserver>
    </applicationserver>
    <database kind="WorklightAdmin" validate="false">
    [...]

<target name="install">
  <configureapplicationserver contextroot="${worklight.contextroot}">
    <project warfile="${worklight.project.war.file}"/>
    <applicationserver>
      <websphereapplicationserver installdir="${appserver.was.installdir}"
                                  profile="${appserver.was.profile}"
                                  libertyEncoding="none">
        <server name="${appserver.was85liberty.serverInstance}"/>
      </websphereapplicationserver>
    </applicationserver>
   <database kind="Worklight" validate="false">
   [...]
    </database>
    <database kind="WorklightReports" validate="false">
   [...]
```

4. Put the encrypted values in the properties for the database passwords.

   For more information about the parameters of Ant tasks, see "Ant tasks for installation of MobileFirst runtime environments" on page 15-32 and "Ant tasks for installation of MobileFirst Operations Console and Administration Services" on page 15-24.

   Note: You can use this `configure-liberty-<`*`database`*`>.xml` file only to deploy, update, or uninstall applications to an application server, with the targets `install`, `minimal-update`, `uninstall`, `adminstal`, `adm-minimalupdate`, or `admuninstall`. You cannot use this file to create or migrate a database, with the targets `databases` or `admdatabases`, because the Ant file must have valid unencrypted database credentials to perform the database operations.

What to do next

See also:

- "Deploying the MobileFirst Operations Console and Administration Services with Ant tasks" on page 6-64

- “Deploying a project WAR file and configuring the application server with Ant tasks” on page 11-16

Ant tasks for installation of MobileFirst Operations Console and Administration Services

The <installworklightadmin>, <updateworklightadmin>, and <uninstallworklightadmin> Ant tasks are provided for the installation of the MobileFirst Operations 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 MobileFirst Operations Console. This task has the following effects:

- It declares the Administration Services web application in the specified context root, by default /worklightadmin.
- For the relational databases, 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 MobileFirst Operations Console as a web application in the specified context root, by default /worklightconsole. If the MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst Operations 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 MobileFirst 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 MobileFirst Operations 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.

Note: On WebSphere Application Server Liberty profile, the task does not change the features, which leaves a potential non-minimal list of features in the `server.xml` file for the installed application.

`<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 MobileFirst Operations Console WAR file from the application server as an option.
- For the relational DBMS, it removes the data sources and – on WebSphere Application Server Full Profile – the JDBC providers for Administration Services.
- For the relational DBMS, 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 15-20. 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 MobileFirst runtime environments, applications, and adapters	No	`/worklightadmin`
id	Distinguishes different deployments	No	Empty
environmentId	Distinguishes different MobileFirst 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 15-20. 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 MobileFirst Operations 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 MobileFirst Server administration and MobileFirst 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 MobileFirst 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 MobileFirst Operations Console. If you set this attribute, you can add the following files to that directory:

- mobilefirst-console.url: This file is a Windows shortcut. It opens the MobileFirst Operations Console in a browser.
- mobilefirst-console.sh: This file is a UNIX shell script and opens the MobileFirst Operations Console in a browser.
- worklight-admin-service.url: This file is a Windows shortcut. It opens in a browser and calls a REST service that returns a list of the MobileFirst projects that can be managed in JSON format. For each listed MobileFirst 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 MobileFirst project runtime is running or idle.
- worklight-admin-service.sh: This file is a UNIX shell script that provides the same output as the worklight-admin-service.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 MobileFirst runtime projects. If MobileFirst 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 15-21. 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 MobileFirst Operations Console

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

Table 15-22. Attributes of the <console> element

Attribute	Description	Required	Default
contextroot	URI of the MobileFirst Operations Console	No	/worklightconsole
install	Indicates whether the MobileFirst Operations 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 15-23. 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 15-24. Attributes for the <property> element

Attribute	Description	Required	Default value
name	Name of the property	Yes	None
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 MobileFirst Operations Console WAR files.

For more information about the JNDI properties, see "List of JNDI properties for MobileFirst Server administration" on page 6-101.

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 Table 15-36 on page 15-35 through Table 15-43 on page 15-37 of "Ant tasks for installation of MobileFirst runtime environments" on page 15-32.

Table 15-25. 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 MobileFirst Server administration and the MobileFirst projects

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

Table 15-26. 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 MobileFirst projects (for Tomcat only)	No	8686

Table 15-26. Attributes of the <jmx> element (continued)

Attribute	Description	Required	Default
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 MobileFirst Server administration and the managed MobileFirst 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 MobileFirst 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 15-27. 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 for relational DBMS, see Table 15-49 on page 15-40 through Table 15-59 on page 15-44 in “Ant tasks for installation of MobileFirst runtime environments” on page 15-32.

Note: The attributes of the Cloudant element are slightly different from the runtime. For more information, see the following table, *Inner elements for the <applicationserver> element*.

Table 15-28. Inner elements for the <database> 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
cloudant	Parameter for Cloudant databases	0..1
driverclasspath	Parameter for JDBC driver class path (relational DBMS only)	0..1

Table 15-29. Attributes of the <cloudant> element

Attribute	Description	Required	Default
url	The URL of the Cloudant account	No	https:// *user*.cloudant.com
user	The user name of the Cloudant account	Yes	None
password	The password of the Cloudant account	No	Queried interactively
dbName	The cloudant database name. **Important:** This database name must start with a lowercase letter and contain only lowercase characters (a-z), Digits (0-9), any of the characters _, $, and -	No	mfp_admin_db

Note: The **<driverclasspath>** element is not relevant to Cloudant, and you do not have to specify it.

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 15-30. 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
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 MobileFirst Operations Console.

- worklightmonitor
- worklightoperator
- worklightdeployer
- worklightadmin

For information about which authorizations are implied by the specific roles, see the chapter about the "REST API Administration Services" on page 10-4.

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

Ant tasks for installation of MobileFirst 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 MobileFirst project WAR file as a web application. This task has the following effects.

- It declares the MobileFirst 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 runtime and reports.
- It deploys the MobileFirst Server runtime file worklight-jee-library.jar and the database drivers in the application server.
- It sets MobileFirst configuration properties through JNDI environment entries. These JNDI environment entries override the MobileFirst 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 on an application server a MobileFirst web application that is already configured. 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 MobileFirst Server runtime worklight-jee-library.jar library file.
- It updates the application server configuration related to BIRT reports, which is a deprecated feature, or, if such reports are no longer used, it sets the value of the JNDI environment entry reports.exportRawData to “false”.

Other than that, the task does not change the application server configuration, that is, the web application configuration, data sources, and JNDI environment entries.

Note: On WebSphere Application Server Liberty profile, the task does not change the features, which leaves a potential non-minimal list of features in the server.xml file for the installed application.

<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 MobileFirst 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 runtime and reports.
- It removes the MobileFirst 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 15-31. 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 MobileFirst environments	No	Empty
wasStartingWeight	Start order for WebSphere Application Server. Lower values start first.	No	2
shortcutsDir	Directory where to place shortcuts	No	None
useWorklightReports	Specifies whether you install the reports database or not. The possible values are `true`, `false`, or `auto`.	No	Auto

`contextroot` and `id`

The `contextroot` and `id` attributes distinguish different MobileFirst projects. By default, when a project is created in V6.0.0 of this product and higher, its context root is the name of the project. The default value of `/worklight` was chosen to facilitate compatibility with IBM 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 MobileFirst Server administration and MobileFirst 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 MobileFirst Administration Services web application is deployed and started before any other MobileFirst runtime projects. If MobileFirst projects are deployed or started before the web application, the JMX communication is not established and you cannot manage your MobileFirst projects.

shortcutsDir

The shortcutsDir attribute for the <unconfigureApplicationServer> Ant task specifies where to expect the shortcuts to the MobileFirst Operations Console if it was installed by a version of the <configureApplicationServer> Ant task older than V6.2.0. 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.

useWorklightReports

If this parameter is set to false, the reports data source is not installed. If set to true, the reports data source is installed. If set to auto, the reports data source is installed only if the task contains a database element of kind WorklightReports. This attribute has no effect on the <updateapplicationserver> task when you upgrade, and the status of the reports remains untouched.

The <configureapplicationserver>, <updateapplicationserver>, and <unconfigureapplicationserver> tasks support the following elements:

Table 15-32. 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 **Note:** The predefined BIRT reports are deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead.	0..1
database	Databases	1..2
analytics	Analytics	0..1

The <project> element specifies details about the project to deploy to the application server. It has the following attributes:

Table 15-33. 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 MobileFirst 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 11-4.

By default, the WAR file is automatically migrated to the current MobileFirst 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 MobileFirst version that produced the WAR file is not suitable for the MobileFirst 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 15-34. 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 MobileFirst properties, or for a list of properties that you can set, see "Configuration of MobileFirst applications on the server" on page 11-50.

The `<applicationserver>` element describes the application server to which the MobileFirst application is deployed. It is a container for one of the following elements:

Table 15-35. 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 later. 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 15-36. 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

Table 15-36. Attributes for the <websphereapplicationserver> or <was> element (continued)

Attribute	Description	Required	Default
libertyEncoding	Algorithm to encode data source passwords for WebSphere Application Server Liberty. The possible values are `none`, `xor`, and `aes`. **Note:** `aes` is not supported for WebSphere Application Server Liberty V8.5.0.x. For the other versions, it is supported only with the default key. On WebSphere Application Server Liberty 8.5.5.x, if the `aes` encoding is requested, the clear password is passed as argument to the `sercurityUtility` program, which is called through an external process. You can see the password with a `ps` command, or in the `/proc` file system on UNIX operating systems.	No	xor
libertyAESKey	Do not set this property because the `aes` custom key is not supported.	No	

It supports the following elements for single-server deployment:

Table 15-37. 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 15-38. 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 15-39. Inner elements for the <was> element (network deployment)

Element	Description	Count
cell	The entire cell	0..1
cluster	All servers of a cluster	0..1
node	All servers in a node, clusters excluded	0..1
server	A single server	0..1

The `<cell>` element has no attributes.

The `<cluster>` element has the following attributes:

Table 15-40. Attributes for the <cluster> element (network deployment)

Attribute	Description	Required	Default
name	Cluster name	Yes	None

The <node> element has the following attributes:

Table 15-41. 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 15-42. Attributes for the <server> element (network deployment)

Attribute	Description	Required	Default
nodeName	Node name	Yes	None
serverName	Server name	Yes	None

The <tomcat> element denotes an Apache Tomcat server. It has the following attributes:

Table 15-43. 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 reports.

The <reports> element has the following attribute:

Note: The predefined BIRT reports are deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead.

Table 15-44. Attributes of the <reports> element

Attribute	Description	Required	Default
todir	Destination directory	Yes	None

The <reports> element supports the following element:

Table 15-45. 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 MobileFirst 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:

Note: The Reports database, referenced as **`WorklightReports`** and **`WLREPORT`** in the subsequent tables, is deprecated in V7.1.0. Use "Operational analytics" on page 13-6 instead. Note that setting up the Reports database is optional in this release and earlier releases.

Table 15-46. Attributes of the <database> element

Attribute	Description	Required	Default
`kind`	The kind of database: **`Worklight`** or **`WorklightReports`**	Yes	None
`validate`	To validate whether the database is accessible or not. The possible values are `true` or `false`.	No	`true`

The `<database>` element supports the following elements:

Table 15-47. 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
`cloudant`	Parameters for Cloudant	0..1
`driverclasspath`	JDBC driver class path **Note:** This element is not relevant to Cloudant	0..1

The `<analytics>` element indicates that you want to connect the MobileFirst runtime to an already installed MobileFirst Operational Analytics console and services. It has the following attributes:

Table 15-48. Attributes of the <analytics> element

Attribute	Description	Required	Default
`install`	Whether to connect the MobileFirst runtime to MobileFirst Operational Analytics	No	`true`
`analyticsURL`	MobileFirst Operational Analytics Services URL	Yes	None
`consoleURL`	MobileFirst Operational Analytics Console URL	Yes	None
`username`	The user name	Yes	None

Table 15-48. Attributes of the <analytics> element (continued)

Attribute	Description	Required	Default
password	The password	Yes	None
validate	To validate whether the MobileFirst Operational Analytics Console is accessible or not.	No	`true`
forwardLogs	Whether to send server logs to MobileFirst Operational Analytics	No	`true`
tenant	The tenant for indexing data that is collected from a MobileFirst runtime.	No	None

install

Use the `install` attribute to indicate that this MobileFirst runtime must be connected and send events to MobileFirst Operational Analytics. Valid values are `true` or `false`.

analyticsURL

Use the `analyticsURL` attribute to specify the URL that is exposed by MobileFirst Operational Analytics, which receives incoming analytics data.

For example: `http://<hostname>:<port>/analytics-service/data`

Note: Optionally, you might append a tenant to this URL. For more information, see "Multi-tenancy" on page 13-19.

consoleURL

Use the `consoleURL` attribute to the URL that is exposed by MobileFirst Operational Analytics, which links to the MobileFirst Operational Analytics console.

For example: `http://<hostname>:<port>/analytics/console`

Note: Optionally, you might append a tenant to this URL. For more information, see "Multi-tenancy" on page 13-19.

username

Use the `username` attribute to specify the user name that is used if the data entry point for the MobileFirst Operational Analytics is protected with basic authentication.

password

Use the `password` attribute to specify the password that is used if the data entry point for the MobileFirst Operational Analytics is protected with basic authentication.

validate

Use the `validate` attribute to validate whether the MobileFirst Operational Analytics Console is accessible or not, and to check the user name authentication with a password. The possible values are `true`, or `false`

forwardLogs

When the `forwardLogs` attribute is set to `true`, server logs that are recorded on the MobileFirst Server are captured and forwarded to the MobileFirst Operational Analytics.

tenant

For more information about this attribute, see “Multi-tenancy” on page 13-19.

To specify an Apache Derby database

The `<derby>` element has the following attributes:

Table 15-49. 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 15-50. 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 IBM MobileFirst Platform Foundation for iOS, a`<driverclasspath>` element is not necessary.

To specify a DB2 database

The `<db2>` element has the following attributes:

Table 15-51. 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

Table 15-51. Attributes of the <db2> element (continued)

Attribute	Description	Required	Default
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 IBM MobileFirst Platform Server runtime operations" on page 6-20.	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 15-52. 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 15-53. 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

Table 15-53. Attributes of the <mysql> element (continued)

Attribute	Description	Required	Default
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 IBM MobileFirst Platform Server runtime operations" on page 6-20.	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 15-54. 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 IBM MobileFirst Platform Server runtime operations" on page 6-20.	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 15-55. 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 15-56. Attributes of the <oracle> element

Attribute	Description	Required	Default
database	Database name, or Oracle service name **Note:** You must always use a service name to connect to a PDB database.	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 IBM MobileFirst Platform Server runtime operations" on page 6-20. See the note under this table.	Yes	None
password	Password for accessing databases	No	Queried interactively

Note: For the **user** attribute, use preferably a user name in uppercase letters. Oracle user names are generally in uppercase letters. Unlike other database tools, the **configureapplicationserver** Ant task does not convert lowercase letters to uppercase letters in the user name. If the **configureapplicationserver** Ant task fails to connect to your database, try to enter the value for the **user** attribute in uppercase letters.

Instead of `database`, `server`, and `port`, you can also specify a URL. In this case, use the following attributes:

Table 15-57. 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 IBM MobileFirst Platform Server runtime operations" on page 6-20. See the note under this table.	Yes	None
password	Password for accessing databases	No	Queried interactively

Note: For the **user** attribute, use preferably a user name in uppercase letters. Oracle user names are generally in uppercase letters. Unlike other database tools, the **configureapplicationserver** Ant task does not convert lowercase letters to uppercase letters in the user name. If the **configureapplicationserver** Ant task fails to connect to your database, try to enter the value for the **user** attribute in uppercase letters.

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 15-58. 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 15-59. 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

To specify a Cloudant database

The `<cloudant>` element has the following attributes:

Table 15-60. Attributes of the <cloudant> element

Attribute	Description	Required	Default
url	The URL of the Cloudant account	No	`https://`*`user`*`.cloudant.com`
user	The user name of the Cloudant account	Yes	None
password	The password of the Cloudant account	No	Queried interactively

Table 15-60. Attributes of the <cloudant> element (continued)

Attribute	Description	Required	Default
`dbNamePrefix`	The prefix that is added to the database name of the MobileFirst runtime. For example, if the value `department` is assigned to this attribute, and the context root of your MobileFirst runtime is `/mfp`, the database name would be `department_mfp_runtime_db`. **Important:** This prefix must start with a lowercase letter, and can contain only lowercase characters (a-z), digits (0-9), and any of the following characters: _, $, -.	No	Empty string
`connectTimeout`	The timeout to establish a connection, in milliseconds. A value of zero means an infinite timeout.	No	Undocumented default value
`socketTimeout`	The timeout to wait for a response, in milliseconds. A value of zero means an infinite timeout.	No	Undocumented default value
`maxConnections`	The maximum number of connections to the database.	No	Undocumented default value
`proxyHost`	The host name to connect through the proxy	No	Undocumented default value
`proxyPort`	The proxy port	No	Undocumented default value

Note: The `<driverclasspath>` element is not relevant to Cloudant, and you do not have to specify it.

Ant tasks for installation of Application Center

The `<installApplicationCenter>`, `<updateApplicationCenter>`, and `<uninstallApplicationCenter>` Ant tasks are provided for the installation of the Application Center Console and Services.

Task effects

`<installApplicationCenter>`

The `<installApplicationCenter>` task configures an application server to run the Application Center Services WAR file as a web application, and to install the Application Center Console. This task has the following effects:

- It declares the Application Center Services web application in the `/applicationcenter` context root.
- It declares data sources, and on WebSphere Application Server full profile, it declares also JDBC providers for Application Center Services.
- It deploys the Application Center Services web application on the application server.
- It declares the Application Center Console as a web application in the `/appcenterconsole` context root.
- It deploys the Application Center Console WAR file on the application server.

- It configures configuration properties for Application Center Services by using JNDI environment entries. The JNDI environment entries that are related to the endpoint and proxies are commented. You must uncomment them in some cases. For more information, see "Defining the endpoint of the MobileFirst Administration services" on page 6-91.
- It configures users that it maps to roles used by the Application Center Console and Services web applications.
- On WebSphere Application Server, it configures the necessary custom property for the web container.

<updateApplicationCenter>

The <updateApplicationCenter> task updates an already configured Application Center application on an application server. This task has the following effects:

- It updates the Application Center Services WAR file. This file must have the same base name as the corresponding WAR file that was previously deployed.
- It updates the Application Center 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, and user-to-role mappings. This task applies only to an installation that is performed by using the <installApplicationCenter> task that is described in this topic.

Note: On WebSphere Application Server Liberty profile, the task does not change the features, which leaves a potential non-minimal list of features in the server.xml file for the installed application.

<uninstallApplicationCenter>

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

- It removes the configuration of the Application Center Services web application with the /applicationcenter context root. As a consequence, the task also removes the settings that were added manually to that application.
- It removes both the Application Center Services and Console WAR files from the application server.
- It removes the data sources and, on WebSphere Application Server full profile, it also removes the JDBC providers for the Application Center Services.
- It removes the database drivers that were used by Application Center Services from the application server.
- It removes the associated JNDI environment entries.
- It removes the users who are configured by the <installApplicationCenter> invocation.

Attributes and elements

The <installApplicationCenter>, <updateApplicationCenter>, and <uninstallApplicationCenter> tasks have the following attributes:

Table 15-61. Attributes for the <installApplicationCenter>, <updateApplicationCenter>, and <uninstallApplicationCenter> Ant tasks

Attribute	Description	Required	Default
id	It distinguishes different deployments in WebSphere Application Server full profile	No	Empty
servicewar	The WAR file for the Application Center Services	No	The `applicationcenter.war` file is in the application Center console directory: *product_install_dir*/`ApplicationCenter/console`.
shortcutsDir	The directory where you place the shortcuts	No	None
aaptDir	The directory that contains the `aapt` program, from the Android SDK platform-tools package.	No	None

id

In WebSphere Application Server full profile environments, the `id` attribute is used to distinguish different deployments of Application Center Console and Services. Without this `id` attribute, two WAR files with the same context roots might conflict and these files would not be deployed.

servicewar

Use the `servicewar` attribute to specify a different directory for the Application Center 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 Application Center Console. If you set this attribute, the following files are added to this directory:

- `appcenter-console.url`: This file is a Windows shortcut. It opens the Application Center Console in a browser.
- `appcenter-console.sh`: This file is a UNIX shell script. It opens the Application Center Console in a browser.

aaptDir

The `aapt` program is part of the IBM MobileFirst Platform Foundation for iOS distribution: *product_install_dir*`/ApplicationCenter/tools/android-sdk`.

If this attribute is not set, during the upload of an `apk` application, Application Center parses it by using its own code, which might have limitations.

The `<installApplicationCenter>`, `<updateApplicationCenter>`, and `<uninstallApplicationCenter>` tasks support the following elements:

Table 15-62. Inner elements for the <installApplicationCenter>, < updateApplicationCenter>, and <uninstallApplicationCenter> Ant tasks

Element	Description	Count
applicationserver	The application server	1
console	The Application Center console	1
database	The databases	1
user	The user to be mapped to a security role	0..∞

To specify an Application Center console

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

Table 15-63. Attributes for the <console> element

Attribute	Description	Required	Default
warfile	The WAR file for the Application Center Console	No	The appcenterconsole.war file is in the Application Center console directory: *product_install_dir*/ ApplicationCenter/ console.

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 Table 15-36 on page 15-35 to Table 15-43 on page 15-37 of the page "Ant tasks for installation of MobileFirst runtime environments" on page 15-32.

Table 15-64. Inner elements for the <applicationserver> element

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

To specify a connection to the services database

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

You must declare a single database: <database kind="ApplicationCenter">. 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 15-65. Attributes for the <database> element

Attribute	Description	Required	Default
kind	The kind of database (ApplicationCenter)	Yes	None
validate	To validate whether the database is accessible or not	No	True

The <database> element supports the following elements. For more information about the configuration of these database elements, see Table 15-49 on page 15-40 to Table 15-59 on page 15-44 in "Ant tasks for installation of MobileFirst runtime environments" on page 15-32

Table 15-66. Inner elements for the <database> element

Element	Description	Count
db2	The parameter for DB2 databases	0..1
derby	The parameter for Apache Derby databases	0..1
mysql	The parameter for MySQL databases	0..1
oracle	The parameter for Oracle databases	0..1
driverclasspath	The 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 15-67. Attributes for the <user> element

Attribute	Description	Required	Default
role	The user role **appcenteradmin**	Yes	None
name	The user name	Yes	None
password	The password, if you must create the user	No	None

This Ant task supports only the **appcenteradmin** role. Users that are defined by using the <user> element can be mapped only to the **appcenteradmin** role for authentication in the Application Center Console.

Ant tasks for installation of MobileFirst Data Proxy

Reference information for the <installdataproxy>, <updatedataproxy>, and <uninstalldataproxy> Ant tasks.

About the Ant tasks for installing MobileFirst Data Proxy

The purpose of these Ant Tasks is to configure the MobileFirst Data Proxy on an application server so that it can communicate with the following components:

- a Cloudant database (either Cloudant Managed or Cloudant Local)
- an Authorization Server that is embedded in a MobileFirst runtime environment.

For a Liberty server, and a stand-alone WebSphere Application Server, these Ant tasks install a Trust Association Interceptor (TAI), which acts as the front end for performing OAuth authentication of incoming requests to the MobileFirst Data Proxy.

Note: None of these Ant tasks install, update, or uninstall a MobileFirst runtime environment.

For more information about how to use these Ant tasks, see “Installing the MobileFirst Data Proxy with Ant tasks” on page 6-201.

For more information about installing a MobileFirst runtime environment, see “Installing the MobileFirst runtime environment” on page 6-114.

Important:

- The MobileFirst Data Proxy is not supported on Apache Tomcat, or on versions of WebSphere Application Server Liberty profile earlier than V8.5.5.0.
- You must install the TAI manually on WebSphere Application Server Network Deployment.

Task effects

`<installdataproxy>`

The `<installdataproxy>` Ant task configures an application server to run a MobileFirst Data Proxy. This task has the following effects:

- It deploys the MobileFirst Data Proxy WAR file on the application server.
- It configures a TAI on WebSphere Application Server Liberty and WebSphere Application Server standalone, and copies the requested files in the application server in the appropriate locations. On WebSphere Application Server standalone, it creates a configuration file for this TAI if that is required.
- It declares the MobileFirst web application in the specified context root, by default `/imf-data-proxy`.
- It sets MobileFirst Data Proxy configuration properties through JNDI environment entries.
- On WebSphere Application Server Liberty, it configures the web container.

`<updatedataproxy>`

The `<updatedataproxy>` Ant task updates the MobileFirst Data Proxy web application, which is already configured, 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 TAI files on WebSphere Application Server Liberty and WebSphere Application Server stand-alone.

The task does not change the application server configuration, which includes the web application configuration, data sources, and JNDI environment entries.

<uninstalldataproxy>

The <uninstalldataproxy> Ant task undoes the effects of an earlier <installdataproxy> run. This task has the following effects.

- It removes the configuration of the MobileFirst Data Proxy web application with the specified context root.
- It removes the MobileFirst Data Proxy WAR file from the application server.
- It removes the associated JNDI environment entries.

Note: It does not remove the TAI JAR and feature manifest files from the application server.

Attributes and elements

The <installdataproxy>, <updatedataproxy>, and <uninstalldataproxy> tasks have the following attributes:

Table 15-68. Attributes for the <installdataproxy>, <updatedataproxy>, and <uninstalldataproxy> Ant tasks

Attribute	Description	Required	Default
contextroot	Common prefix in URLs to the application (context root)	No	/imf-data-proxy
id	Distinguishes different deployments	No	Empty
serviceWar	The WAR file for the MobileFirst Data Proxy.	No	The imf-data-proxy WAR file is in the directory Datastore, at the same level than the WorklightServer directory.
DBAccount	Cloudant database account, or host name of the MobileFirst Cloudant Local server.	Yes	None
DBAccountUser	User name for accessing the database.	Yes	None
DBAccountPassword	Password for accessing the database.	Yes	Queried interactively
validate	To validate whether the Cloudant database is accessible or not. The possible values are true or false.	No	true
cloudantPort	Port to access Cloudant.	No	443 or 80
useHttps	To indicate whether the access to Cloudant must use https. The possible values are true or false.	No	true

contextroot and id

The contextroot and id attributes distinguish different MobileFirst projects.

In WebSphere Application Server Liberty profiles, the contextroot parameter is sufficient for this purpose. In WebSphere Application Server full profile environments, the id attribute is used instead.

serviceWar

Use the serviceWar attribute to specify a different directory for the MobileFirst Data Proxy WAR file. You can specify the name of this WAR file with an absolute path or a relative path.

DBAccount

The DBAccount attribute identifies either an Cloudant Managed account name or a user name defined in the configuration of a MobileFirst Cloudant Local database.

DBAccountUser

Use the DBAccountUser attribute to specify the name of a user to connect to Cloudant.

This user name can be an administrator or a non-administrator user. In all cases, the user name you provide must exist in Cloudant as the Ant task does not create it.

DBAccountPassword

Use the DBAccountPassword attribute to specify the password associated to the user specified with attribute DBAccountUser.

validate

When set to true, the validate attribute enables you to check that the Cloudant database identified by attribute DBAccount is on a reachable system and can be accessed using the credentials provided with attributes DBAcountUser and DBAcountPassword. By default, this validation is enabled.

cloudantPort

With the cloudantPort attribute, you can specify a port to access a Cloudant database directly, or a Load Balancer port if a Cloudant haproxy service is used to access the Cloudant database. If this attribute is not specified, the default port is set to 443, unless the useHttps attribute is set to false. In this case, the default port is set to 80.

useHttps

With the useHttps attribute, you can indicate which of the http or https protocol the MobileFirst Data Proxy must use to connect to the Cloudant database. The default protocol used is https.

The <installdataproxy>, <updatedataproxy>, and <uninstalldataproxy> tasks support the following elements:

Table 15-69. Inner elements for the <installdataproxy>, <updatedataproxy>, and <uninstalldataproxy> Ant tasks

Element	Description	Count
authenticate	TAI	0 for WebSphere Application Server Network Deployment 1 for WebSphere Application Server Liberty profile and WebSphere Application Server full profile
applicationserver	application server	1

Important: You cannot specify an **authenticate** element for WebSphere Application Server Network Deployment environments.

To install a TAI

Use the <authentication> element to define the TAI and, optionally, the location of the files that implement it. The attributes of this element are described in Table 3.

Table 15-70. Attributes for the <authentication> element

Attribute	Description	Count	Required	Default
serverURL	The Authorization Server URL	1	Yes	None
taiJAR	The TAI JAR file	0..1	No	File com.ibm.worklight.oauth.tai_1.0.0.jar located in a subdirectory external-server-libraries of the directory where you find worklight-ant-deployer.jar
libertyFeatureManifest	The TAI feature manifest (for Liberty only)	0..1	No	File OauthTai-1.0.mf located in a subdirectory external-server-libraries of the worklight-ant-deployer.jar location

Note: The tairJAR and libertyFeatureManifest attributes are not mandatory. For WebSphere Application Server full profile environments, you might want to specify a taiJAR attribute. For Liberty profile environments, if you specify one of these attributes, you must also specify the other.

serverURL

Use the serverURL attribute to specify the URL of a MobileFirst project in the form *http[s]*://hostname:*port*/context-root, where <context-root> is the context root of a MobileFirst project.

taiJAR

Use the taiJAR attribute to specify a different directory for the MobileFirst Data Proxy TAI JAR file. You can specify the name of this JAR file with an absolute path or a relative path. By default, the Ant task uses the file com.ibm.worklight.oauth.tai_1.0.0.jar that is in a subdirectory external-server-libraries of the directory where worklight-ant-deployer.jar is.

libertyFeatureManifest

Use the libertyFeatureManifest attribute to specify a different directory for the Liberty feature manifest file. You can specify the name of this file with an absolute path or a relative path. By default, the Ant task uses the file OauthTai-1.0.mf that is in a subdirectory external-server-libraries of the worklight-ant-deployer.jar location.

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 element. The attributes and inner elements of this element are described in tables 6 through 12 of "Ant tasks for installation of MobileFirst runtime environments" on page 15-32.

Note: You cannot specify Apache Tomcat, as this application server is not supported by MobileFirst Data Proxy.

Table 15-71. Inner elements for the <applicationserver> element.

Element	Description	Count
websphereapplicationserver or **was**	The parameters for WebSphere Application Server.	1

Ant tasks for installation of MobileFirst Operational Analytics

The `<installanalytics>`, `<updateanalytics>`, and `<uninstallanalytics>` Ant tasks are provided for the installation of MobileFirst Operational Analytics.

About these Ant tasks

The purpose of these Ant Tasks is to configure the MobileFirst Operational Analytics console and the MobileFirst Operational Analytics service with the appropriate storage for the data on an application server.

The task installs MobileFirst Operational Analytics nodes that act as a master and data. For more information, see "Setting up a production cluster" on page 13-85.

Task effects

`<installanalytics>`

The `<installanalytics>` Ant task configures an application server to run IBM MobileFirst Platform Operational Analytics. This task has the following effects:

- It deploys the MobileFirst Operational Analytics Service and the MobileFirst Operational Analytics Console WAR files on the application server.
- It declares the MobileFirst Operational Analytics Service web application in the specified context root `/analytics-service`.
- It declares the MobileFirst Operational Analytics Console web application in the specified context root `/analytics`.
- It sets MobileFirst Operational Analytics Console and MobileFirst Operational Analytics Services configuration properties through JNDI environment entries.
- On WebSphere Application Server Liberty profile, it configures the web container.
- Optionally, it creates users to use the MobileFirst Operational Analytics Console.

`<updateanalytics>`

The `<updateanalytics>` Ant task updates the already configured MobileFirst Operational Analytics Service and MobileFirst Operational Analytics Console web applications WAR files on an application server. These files must have the same base names as the project WAR files that were previously deployed.

The task does not change the application server configuration, that is, the web application configuration and JNDI environment entries.

<uninstallanalytics>

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

- It removes the configuration of both the MobileFirst Operational Analytics Service and the MobileFirst Operational Analytics Console web applications with their respective context roots.
- It removes the MobileFirst Operational Analytics Service and the MobileFirst Operational Analytics Console WAR files from the application server.
- It removes the associated JNDI environment entries.

Attributes and elements

The <installanalytics>, <updateanalytics>, and <uninstallanalytics> tasks have the following attributes:

Table 15-72. Attributes for the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks

Attribute	Description	Required	Default
serviceWar	The WAR file for the MobileFirst Operational Analytics Service	No	The analytics-service.war file is in the directory Analytics.

serviceWar

Use the serviceWar attribute to specify a different directory for the MobileFirst Operational Analytics Services WAR file. You can specify the name of this WAR file with an absolute path or a relative path.

The <installanalytics>, <updateanalytics>, and <uninstallanalytics> tasks support the following elements:

Table 15-73. Inner elements for the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks

Attribute	Description	Required	Default
console	MobileFirst Operational Analytics	Yes	1
user	The user to be mapped to a security role.	No	0..∞
storage	The type of storage.	Yes	1
applicationserver	The application server.	Yes	1
property	Properties.	No	0..∞

To specify a MobileFirst Operational Analytics Console

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

Table 15-74. Attributes of the <console> element

Attribute	Description	Required	Default
warfile	The console WAR file	No	The analytics-ui.war file is in the Analytics directory.
shortcutsdir	The directory where you place the shortcuts.	No	None

warFile

Use the warFile attribute to specify a different directory for the MobileFirst Operational Analytics Console 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 MobileFirst Operational Analytics Console. If you set this attribute, you can add the following files to that directory:

- analytics-console.url: This file is a Windows shortcut. It opens the MobileFirst Operational Analytics Console in a browser.
- analytics-console.sh: This file is a UNIX shell script. It opens the MobileFirst Operational Analytics Console in a browser.

Note: These shortcuts do not include the ElasticSearch tenant parameter.

The <console> element supports the following nested element:

Table 15-75. Inner element of the <console> element

Element	Description	Count
property	Properties	0..∞

With this element, you can define your own JNDI properties.

The <property> element has the following attributes:

Table 15-76. Attributes of the <property> element

Attribute	Description	Required	Default
name	The name of the property.	Yes	None
value	The value of the property.	Yes	None

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 15-77. 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
password	The password, if the user must 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 MobileFirst Operations Console:

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

To specify a type of storage for MobileFirst Operational Analytics

The `<storage>` element indicates which underlying type of storage MobileFirst Operational Analytics uses to store the information and data it collects.

It supports the following element:

Table 15-78. Inner element of the <storage> element

Element	Description	Count
`elasticsearch`	ElasticSearch cluster	1

The `<elasticsearch>` element collects the parameters about an ElasticSearch cluster.

Table 15-79. Attributes of the <elasticsearch> element

Attribute	Description	Required	Default
`clusterName`	The ElasticSearch cluster name.	No	`worklight`
`nodeName`	The ElasticSearch node name. This name must be unique in an ElasticSearch cluster.	No	`worklightNode_<`*`random number`*`>`
`mastersList`	A comma-delimited string that contains the host name and ports of the ElasticSearch master nodes in the ElasticSearch cluster (For example: `hostname1:transport-port1,hostname2:transport-port2`)	No	Depends on the topology
`dataPath`	The ElasticSearch cluster location.	No	Depends on the application server
`shards`	The number of shards that the ElasticSearch cluster creates. This value can be set only by the master nodes that are created in the ElasticSearch cluster.	No	5
`replicasPerShard`	The number of replicas for each shard in the ElasticSearch cluster. This value can be set only by the master nodes that are created in the ElasticSearch cluster.	No	1
`transportPort`	The port used for node-to-node communication in the ElasticSearch cluster.	No	9600

`clusterName`

Use the `clusterName` attribute to specify a name of your choice for the ElasticSearch cluster.

An ElasticSearch cluster consists of one or more nodes that share the same cluster name so you might specify the same value for the clusterName attribute if you configure several nodes.

nodeName

Use the nodeName attribute to specify a name of your choice for the node to configure in the ElasticSearch cluster. Each node name must be unique in the ElasticSearch cluster even if nodes span on several machines.

mastersList

Use the mastersList attribute to provide a comma-separated list of the master nodes in your ElasticSearch cluster. Each master node in this list must be identified by its host name, and the ElasticSearch node-to-node communication port. This port is 9600 by default, or it is the port number that you specified with the attribute transportPort when you configured that master node.

For example: hostname1:transport-port1, hostname2:transport-port2.

Note:

- If you specify a transportPort that is different than the default value 9600, you must also set this value with the attribute transportPort. By default, when the attribute mastersList is omitted, an attempt is made to detect the host name and the ElasticSearch transport port on all supported application servers.
- If the target application server is WebSphere Application Server Network Deployment cluster, and if you add or remove a server from this cluster at a later point in time, you must edit this list manually to keep in sync with the ElasticSearch cluster.

dataPath

Use the dataPath attribute to specify a different directory to store ElasticsSearch data. You can specify an absolute path or a relative path.

If the attribute dataPath is not specified, then ElasticSearch cluster data is stored in a default directory that is called analyticsData, whose location depends on the application server:

- For WebSphere Application Server Liberty profile, the location is ${wlp.user.dir}/servers/serverName/analyticsData.
- For Apache Tomcat, the location is ${CATALINA_HOME}/bin/analyticsData.
- For WebSphere Application Server and WebSphere Application Server Network Deployment, the location is ${was.install.root}/profiles/<profileName>/analyticsData.

The directory analyticsData and the hierarchy of sub-directories and files that it contains are automatically created at run time, if they do not already exist when the MobileFirst Operational Analytics Service component receives events.

shards

Use the shards attribute to specify the number of shards to create in the ElasticSearch cluster.

replicasPerShard

Use the replicasPerShard attribute to specify the number of replicas to create for each shard in the ElasticSearch cluster.

Each shard can have zero or more replicas. By default, each shard has one replica, but the number of replicas can be changed dynamically on an existing index in the MobileFirst Operational Analytics. A replica shard can never be started on the same node as its shard.

transportPort

Use the transportPort attribute to specify a port that other nodes in the ElasticSearch cluster must use when communicating with this node. You must ensure that this port is available and accessible if this node is behind a proxy or firewall.

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.

Note: The attributes and inner elements of this element are described in tables 6 through 12 of "Ant tasks for installation of MobileFirst runtime environments" on page 15-32.

Table 15-80. Inner elements of the <applicationserver> element

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

To specify custom JNDI properties

The <installanalytics>, <updateanalytics>, and <uninstallanalytics> elements support the following element:

Table 15-81. Inner element of the <property> element

Element	Description	Count
property	Properties	0..∞

By using this element, you can define your own JDNI properties.

This element has the following attributes:

Table 15-82. Attributes of the <property> element

Attribute	Description	Required	Default
name	The name of the property.	Yes	None
value	The value of the property.	Yes	None

Internal runtime databases

You can access common runtime tables in both relational databases and the Cloudant database. In relational databases, the entities are organized in database tables; in a Cloudant database, they are organized as document entities.

The following table provides a list of common runtime database tables and entities, their descriptions, and how they are used in relational databases and the Cloudant database.

Entity name	Description	Relational database table name	Cloudant database mfpEntityType	Order of magnitude (rows for relational databases and documents for Cloudant)
AttributeStoreEntity.	Stores the attribute store in the authorization server that is used for storing the authentication context of the client.	`ATTRIBUTE_STORE`	AttributeStoreEntity	One row or document per application.
ClientInstance	Stores instances of client applications that have registered with the OAuth server.	`CLIENT_INSTANCES`	ClientInstance	One row or document per device or application pair.
ClusterSingletonDetails.	Internal cluster synchronization tasks.	`CLUSTER_SYNC`	ClusterSingletonDetails	Tens of rows or documents.
Device	Tracks Client Devices that access the platform. The Client Devices are recorded as active or inactive, based on the configured decommissioning policy, and other information might be stored for them.	`DEVICES`	Device	One row or document per device that accessed 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.wh`** parameters.
GadgetDeviceAssociation	Stores relationships between a device and the applications that the device uses.	`GADGET_DEVICE_ASSOC`	GadgetDeviceAssociation	One row or document per device or application pair.

Entity name	**Description**	**Relational database table name**	**Cloudant database mfpEntityType**	**Order of magnitude (rows for relational databases and documents for Cloudant)**
UserPref	User preferences according to unique user identifier. No user preferences are ready for immediate use. The App developer can add preferences.	`GADGET_USER_PREF`	UserPref	If used, this table can contain one row or document per preference per user.
LicenseTerms	Stores the various license metrics captured every time the device decommissioning task is run.	`LICENSE_TERMS`	LicenseTerms	Tens of rows or documents. This value does not exceed the value set by the **`wl.device.decommission.when`** property.
PushDevice	Push notification table. Stores a record per device.	`PUSH_DEVICES`	PushDevice	One row or document per device.
PushSubscription	Push notification table. Stores a record per tag subscription or user subscription to event sources.	`PUSH_SUBSCRIPTIONS`	PushSubscription	One row or document per device subscription.
SsoEntity	Stores the active sessions that use the SSO feature.	`SSO_LOGIN_CONTEXTS`	SsoEntity	If SSO enabled, there is one row or document per session.
LicenseTermsAddressableDevice	Stores the addressable device metrics daily. An entry is also added each time that a cluster is started.	ADDRESSABLE_DEVICE	LicenseTermsAddressableDevice	About 400 rows. Entries older than 13 months are deleted daily.
Version	The product version.	`WORKLIGHT_VERSION`	-	One row.

The following table provides a list of common reports database tables and their usage.

Note:

- The Reports database and all the tables in the following list are deprecated in IBM MobileFirst Platform Foundation for iOS V7.1.0. Use “Operational analytics” on page 13-6 instead.
- Setting up the Reports database is optional in this release and earlier releases.
- The Reports database is not supported in Cloudant.

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 the product aggregation task or by the customer aggregation task. For more information about using the row data, see “Using raw data reports” on page 13-98.	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 MobileFirst 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 13-102.	Size depends on app/device usage.

Name	Description	Order of Magnitude
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.	One row per notification.
NOTIFICATION_REPORT	Each time the data processing is done, a time stamp is added to the PROC_REPORT table with the processing result (time stamp 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.

You can access common administration database tables in both relational databases and the Cloudant database. In relational databases, the entities are organized in database tables; in Cloudant, they are organized as document entities. The following table provides a list of common administration database tables and entities, their descriptions, and how they are used in relational databases and the Cloudant database. Entity names in brackets such as “(Version)”, are database tables that occur internally and do not correspond to any Java Entity or Cloudant document entity.

Entity name	Description	Relational database table name	Cloudant database mfpEntityType	Order of Magnitude (rows for relational databases and documents for Cloudant)
AdapterEntity	Stores the adapter deployable elements. This table is used to synchronize the adapter deployable elements between cluster nodes; many-to-many relationships with ProjectEntity.	`ADAPTERS`	Adapter	Tens of rows or documents.
AdminNodeEntity	Stores information about the servers that run the administration services. In a stand-alone topology with only one server, this entity is not used.	`ADMIN_NODE`	AdminNode	One row or document per server; empty if a stand-alone server is used.
ApplicationEntity	Stores the application deployable elements. This table is used to synchronize the application deployable elements between cluster nodes; many-to-many relationships with ProjectEntity.	`APPLICATIONS`	Application	Tens of rows or documents.
ApplicationEnvironmentEntity	Environments, for example, iPhone or Android, of deployed applications. Many-to-one relationships with ApplicationEntity.	`APPLICATIONS_ENVIRONMENTS`	ApplicationEnvironment	Tens of rows or documents.

Entity name	Description	Relational database table name	Cloudant database mfpEntityType	Order of Magnitude (rows for relational databases and documents for Cloudant)
ApplicationVersionAccessDataEntity	Stores the applications that have the remote disable mode to block or notify. References ApplicationEnvironmentEntity.	`APP_VERSION_ACCESS_DATA`	ApplicationVersionAccessData	Tens of rows or documents.
AuditEntity	Stores an audit trail of all administrative actions performed on the administration server.	`AUDIT_TRAIL`	Audit	Thousands of rows or documents.
BeaconEntity	Stores information about registered beacons.	`BEACONS`	Beacon	One row or document per registered beacon.
BeaconTriggerEntity	Stores information about the action that is triggered when the user's mobile device comes into the vicinity of an associated beacon.	`BEACON_TRIGGERS`	BeaconTrigger	Tens of rows or documents.
BeaconTriggerAssociationEntity	Stores the association between beacons and triggers; many-to-many relationship between beacons and beacon triggers.	`BEACON_TRIGGER_ASSOC`	BeaconTriggerAssociation	One row or document for each association between a registered beacon and a beacon trigger.
ClientConfigProfileEntity	Stores custom logging configuration profiles that have been created by the server administrator.	`CONFIG_PROFILES`	ClientConfigProfile	One row or document per configuration profile. Usually no more than 5-10 rows in total.
DifferentialDirectUpdateEntity	Stores information on differential direct updates and their binaries.	`DIFFERENTIAL_DIRECT_UPDATE`	DifferentialDirectUpdate	One row or document per environment per deployment.

Entity name	Description	Relational database table name	Cloudant database mfpEntityType	Order of Magnitude (rows for relational databases and documents for Cloudant)
FarmConfigEntity	Stores the configuration of farm nodes when a server farm is used.	`FARM_CONFIG`	FarmConfig	Tens of rows or documents; empty if no server farm is used.
ProjectEntity	Stores the names of the deployed projects.	`PROJECT`	Project	Tens of rows or documents.
(Project-Adapter-Association)	Bidirectional association between projects and adapters.	`PROJECT_ADAPTERS`		Tens of rows.
(Project-Application-Association)	Bidirectional association between projects and applications.	`PROJECT_APPLICATIONS`		Tens of rows.
ProjectLockEntity	Internal cluster synchronization tasks.	`PROJECT_LOCK`		Tens of rows.
PushEnvironmentEntity	Push notification table; stores details of push environments.	`PUSH_ENVIRONMENTS`	PushEnvironment	Tens of rows or documents.
PushTagEntity	Push notification table; stores details of defined tags.	`PUSH_TAGS`	PushTag	Tens of rows or documents.
TransactionEntity	Internal cluster synchronization table; stores the state of all current administrative actions.	`TRANSACTIONS`	Transaction	Tens of rows or documents.
(Version)	The product version.	`WORKLIGHTMGT_VERSION`		One row.

Sample configuration files

IBM MobileFirst Platform Foundation for iOS includes a number of sample configuration files to help you get started with the Ant tasks to install the MobileFirst Server administration and the MobileFirst 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 MobileFirst Server distribution.

Step 1

Pick the appropriate sample configuration file. The following files are provided

Table 15-83. Sample configuration files provided with IBM MobileFirst Platform Foundation for iOS

Task	Derby	DB2	MySQL	Oracle	Cloudant
Create databases with database administrator credentials	`create-database-derby.xml`	`create-database-db2.xml`	`create-database-mysql.xml`	`create-database-oracle.xml`	Not relevant for this database
Install MobileFirst Server administration and MobileFirst runtime environment on Liberty	`configure-liberty-derby.xml`	`configure-liberty-db2.xml`	`configure-liberty-mysql.xml` (See Note on MySQL)	`configure-liberty-oracle.xml`	`configure-liberty-cloudant.xml`
Install MobileFirst Server administration and MobileFirst runtime environment on WebSphere Application Server full profile, single server	`configure-was-derby.xml`	`configure-was-db2.xml`	`configure-was-mysql.xml` (See Note on MySQL)	`configure-was-oracle.xml`	`configure-was-cloudant.xml`

Install MobileFirst Server administration and MobileFirst runtime environment on WebSphere Application Server Network Deployment
(See Note on configuration files)
	`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 on MySQL)
			`configure-wasnd-server-mysql.xml` (See Note on MySQL)
			`configure-wasnd-node-mysql.xml` (See Note on MySQL)
			`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`
					`configure-wasnd-cluster-cloudant.xml`
					`configure-wasnd-server-cloudant.xml`
					`configure-wasnd-node-cloudant.xml`
					`configure-wasnd-cell-cloudant.xml`

Table 15-83. Sample configuration files provided with IBM MobileFirst Platform Foundation for iOS (continued)

Task	Derby	DB2	MySQL	Oracle	Cloudant
Install MobileFirst Server administration and MobileFirst runtime environment on Apache Tomcat	`configure-tomcat-derby.xml`	`configure-tomcat-db2.xml`	`configure-tomcat-mysql.xml`	`configure-tomcat-oracle.xml`	`configure-tomcat-cloudant.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 on MySQL)	`redeploy506-liberty-oracle.xml`	None
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 on MySQL)	`redeploy506-was-oracle.xml`	None

Install or upgrade IBM Worklight V5.0.6 on WebSphere Application Server Network Deployment
(See Note on configuration files)
	`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 on MySQL)
			`redeploy506-wasnd-server-mysql.xml` (See Note on MySQL)
			`redeploy506-wasnd-node-mysql.xml` (See Note on MySQL)
			`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`
					None

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`	None

Note on MySQL: : 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.

Note on configuration files for WebSphere Application Server Network Deployment: The configuration files for `wasnd` contain a scope that can be set to **`cluster`**, **`node`**, **`server`**, or **`cell`**. For example, for `configure-wasnd-cluster-derby.xml`, the scope is **`cluster`**. These scope types define the deployment target as follows:

- **`cluster`**: To deploy to a cluster.
- **`server`**: To deploy to a single server that is managed by the deployment manager.
- **`node`**: To deploy to all the servers that are running on a node, but that do not belong to a cluster.
- **`cell`**: To deploy to all the servers on a cell.

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 beginning of the file.

Note: The following special characters need to be escaped when used in values in Ant XML scripts:

- The dollar sign ($) must be written as `$$`, unless you explicitly want to reference an Ant variable through the syntax `${variable}`, as described in Properties in the *Apache Ant Manual*.
- The ampersand character (&) must be written as `&amp;`, unless you explicitly want to reference an XML entity.

- Double quotation marks (") must be written as &quot;, except when inside a string that is enclosed in single quotation marks.

Step 5

In the <configureapplicationserver> and <unconfigureapplicationserver> invocations (in target install and uninstall), define MobileFirst properties. For a list of properties that can be set, see "Configuration of MobileFirst applications on the server" on page 11-50. In production, you must often define the following specific properties:

- publicWorkLightHostname
- publicWorkLightProtocol
- publicWorkLightPort

Step 6

Note: You must run the following commands only if you are using a relational database. If you are using Cloudant, you must not run them.

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 MobileFirst Platform Foundation for iOS. 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 runtime database to the administration database, which must be initialized first by the target admdatabases.

For an initial installation, and if a DBA did not create the databases manually, use the file in row "Create databases with database administrator credentials" of Table 15-83 on page 15-67. 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 MobileFirst Operations Console components onto the application server.

To install updated Administration Services and MobileFirst Operations Console components (for example, to apply a MobileFirst 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 MobileFirst Operations Console components.

Step 8

Run the command:

```
ant -f configure-file.xml install
```

This command installs your MobileFirst runtime environment as a .war file onto the application server.

To install an updated MobileFirst 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 MobileFirst 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 MobileFirst project's .war file. This file makes it possible to redeploy an updated .war file with the same MobileFirst 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.

Sample configuration files for MobileFirst Operational Analytics

IBM MobileFirst Platform Foundation for iOS includes a number of sample configuration files to help you get started with the Ant tasks to install the MobileFirst Operational Analytics Services, and the MobileFirst Operational Analytics Console.

The easiest way to get started with the <installanalytics>, <updateanalytics>, and <uninstallanalytics> Ant tasks is by working with the sample configuration files provided in the Analytics/configuration-samples/ directory of the MobileFirst Server distribution.

Step 1

Pick the appropriate sample configuration file. The following XML files are provided. They are referred to as configure-file.xml in the next steps.

Table 15-84. Sample configuration files provided with IBM MobileFirst Platform Foundation for iOS

Task	Application server
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server Liberty profile	configure-liberty-analytics.xml
Install MobileFirst Operational Analytics Services and Console on Apache Tomcat	configure-tomcat-analytics.xml
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server full profile	configure-was-analytics.xml

Table 15-84. Sample configuration files provided with IBM MobileFirst Platform Foundation for iOS (continued)

Task	Application server
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server Network Deployment, single server	`configure-wasnd-server-analytics.xml`
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server Network Deployment, cell	`configure-wasnd-cell-analytics.xml`
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server Network Deployment, node	`configure-wasnd-node.xml`
Install MobileFirst Operational Analytics Services and Console on WebSphere Application Server Network Deployment, cluster	`configure-wasnd-cluster-analytics.xml`

Note on configuration files for WebSphere Application Server Network Deployment:

The configuration files for `wasnd` contain a scope that can be set to **`cluster`**, **`node`**, **`server`**, or **`cell`**. For example, for `configure-wasnd-cluster-analytics.xml`, the scope is **`cluster`**. These scope types define the deployment target as follows:

- **`cluster`**: To deploy to a cluster.
- **`server`**: To deploy to a single server that is managed by the deployment manager.
- **`node`**: To deploy to all the servers that are running on a node, but that do not belong to a cluster.
- **`cell`**: To deploy to all the servers on a cell.

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 your application server is WebSphere Application Server Liberty profile, or Apache Tomcat, 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 files:

- 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 must be escaped when they are used in the values of the 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 section of the Apache Ant Manual.
- The ampersand character (`&`) must be written as `&amp;`, unless you explicitly want to reference an XML entity.
- Double quotation marks (`"`) must be written as `&quot;`, except when it is inside a string that is enclosed in single quotation marks.

Step 5

Run the command:

```
ant -f configure-file.xml install
```

This command installs your MobileFirst Operational Analytics Services and MobileFirst Operational Analytics Console components in the application server.

To install updated MobileFirst Operational Analytics Services and MobileFirst Operational Analytics Console components, for example if you apply a MobileFirst Server fix pack, run the following 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 MobileFirst Operational Analytics Services and MobileFirst Operational Analytics Console components.

Glossary

This glossary provides terms and definitions for IBM MobileFirst Platform Foundation software and products.

The following cross-references are used in this glossary:

- *See* refers you from a nonpreferred term to the preferred term or from an abbreviation to the spelled-out form.
- *See also* refers you to a related or contrasting term.

For other terms and definitions, see the IBM Terminology website (opens in new window).

“A” “B” on page 16-2 “C” on page 16-2 “D” on page 16-4 “E” on page 16-4 “F” on page 16-4 “G” on page 16-5 “H” on page 16-5 “I” on page 16-5 “J” on page 16-5 “K” on page 16-6 “L” on page 16-6 “M” on page 16-7 “N” on page 16-7 “O” on page 16-8 “P” on page 16-8 “R” on page 16-9 “S” on page 16-9 “T” on page 16-10 “U” on page 16-11 “V” on page 16-11 “W” on page 16-11 “X” on page 16-11

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 MobileFirst application. Adapters connect to enterprise applications, deliver data to and from mobile applications, and perform any necessary server-side logic on sent data.

administration database
The database of the MobileFirst Operations Console and of the Administration Services. The database tables define elements such as applications, adapters, projects with their descriptions and orders of magnitude.

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 web or mobile device application. See also web application.

Application Center
A MobileFirst 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 so that one can 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.

authentication realm
A combination of one authenticator and one login module. Each authentication realm defines its authentication flow. An authentication realm must have a corresponding challenge handler.

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 MobileFirst 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.

device enrollment
The process of a device owner registering their device as trusted.

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.

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.

geocoding
The process of identifying geocodes from more traditional geographic markers (addresses, postal codes, and so on). For example, a landmark can be located at the intersection of two streets, but the geocode of that landmark consists of a number sequence. See also geolocation.

geofence
A circle or a polygon that defines a geographical area.

geolocation
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 geocoding, location services.

H

homogeneous server farm
A server farm in which all application servers are of the same type, level, and version.

hybrid application
An application that is primarily written in web-oriented languages (HTML5, CSS, and JS), but is wrapped in a native shell so that the app behaves like, and provides the user with all the capabilities of, a native app.

I

in-house application
See company application.

inner application
An application that contains the HTML, CSS, and JavaScript parts that run within a shell component. Inner applications must be packaged within a shell component to create a full hybrid application.

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.

keychain
A password management system for Apple software. A keychain acts as a secure storage container for passwords that are used by multiple applications and services.

key pair
In computer security, a public key and a private key. When the key pair is used for encryption, the sender uses the receiver's public key to encrypt the message, and the recipient uses their private key to decrypt the message. When the key pair is used for signing, the signer uses their private key to encrypt a representation of the message, and the recipient uses the sender's 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 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 geolocation.

login module
A server-side entity that is responsible for verifying the user credentials and for creating a user identity object that holds the user properties for the remainder of the session.

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.

MobileFirst adapter
See adapter.

MobileFirst Data Proxy
A server-side component to the IMFData SDK that can be used to secure mobile application calls to Cloudant by using MobileFirst Platform OAuth security capabilities. The MobileFirst Data Proxy requires an authentication through the trust association interceptor.

MobileFirst Operations Console
A web-based interface that is used to control and manage MobileFirst runtime environments that are deployed in MobileFirst Server, and to collect and analyze user statistics.

MobileFirst 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).

MobileFirst Server
A MobileFirst component that handles security, back-end connections, push notifications, mobile application management, and analytics. The MobileFirst Server is a collection of apps that run on an application server and acts as a runtime container for MobileFirst runtime environments.

MobileFirst Studio
A MobileFirst component that is an integrated development environment (IDE) that can be used to develop and test mobile applications.

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).

O

OAuth
An HTTP-based authorization protocol that gives applications scoped access to a protected resource on behalf of the resource owner, by creating an approval interaction between the resource owner, client, and resource server.

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 MobileFirst 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 MobileFirst 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.

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

salt Randomly generated data that is inserted into a password or passphrase hash, making those passwords uncommon (and more difficult to hack).

SDK See software development kit.

security test
An ordered set of authentication realms that is used to protect a resource such as an adapter procedure, an application, or a static URL.

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.

sideloading
On Windows 8 environments, the process of loading a file of type appx on a mobile device without using the Windows Store.

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 an Android 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.

software development kit (SDK)
A set of tools, APIs, and documentation to assist with the development of software in a specific computer language or for a particular operating environment.

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.

TAI See trust association interceptor.

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.

trust association interceptor (TAI)
The mechanism by which trust is validated in the product environment for every request received by the proxy server. The method of validation is agreed upon by the proxy server and the interceptor.

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 app
See web application.

web application (web app)
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. See also app.

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.

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.

Support and comments

For the entire IBM MobileFirst Platform 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.

If you would like a response from IBM, please provide the following information:

- Name
- Address
- Company or Organization
- Phone No.
- Email address

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.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

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.

Index